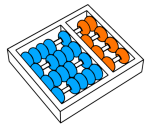
Universidade Estadual de Campinas
Instituto de Computação



Carla Negri Lintzmayer

The Problem of Sorting Permutations by Prefix and Suffix Rearrangements

O Problema da Ordenação de Permutações Usando Rearranjos de Prefixos e Sufixos

CAMPINAS
2016

Carla Negri Lintzmayer

The Problem of Sorting Permutations by Prefix and Suffix Rearrangements

O Problema da Ordenação de Permutações Usando Rearranjos de Prefixos e Sufixos

Tese apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Doutora em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Supervisor/Orientador: Prof. Dr. Zanoni Dias

Este exemplar corresponde à versão final da Tese defendida por Carla Negri Lintzmayer e orientada pelo Prof. Dr. Zanoni Dias.

CAMPINAS
2016

Agência(s) de fomento e nº(s) de processo(s): FAPESP, 2013/01172-0; CNPq, 140017/2013-5

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

L658p

Lintzmayer, Carla Negri, 1990-
The problem of sorting permutations by prefix and suffix rearrangements / Carla Negri Lintzmayer. – Campinas, SP : [s.n.], 2016.

Orientador: Zanoni Dias.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Rearranjo de genomas. 2. Biologia computacional. 3. Ordenação (Computadores). 4. Permutações (Matemática). 5. Algoritmos de aproximação. I. Dias, Zanoni,1975-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: O problema da ordenação de permutações usando rearranjos de prefixos e sufixos
Palavras-chave em inglês:
Genome rearrangements
Computational biology
Sorting (Electronic computers)
Permutations
Approximation algorithms
Área de concentração: Ciência da Computação
Titulação: Doutora em Ciência da Computação
Banca examinadora:
Zanoni Dias [Orientador]
Maria Emília Machado Telles Walter
Cristina Gomes Fernandes
Eduardo Candido Xavier
Fábio Luiz Usberti
Data de defesa: 15-12-2016
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Carla Negri Lintzmayer

The Problem of Sorting Permutations by Prefix and Suffix Rearrangements

O Problema da Ordenação de Permutações Usando Rearranjos de Prefixos e Sufixos

Banca Examinadora:

- Prof. Dr. Zanoni Dias
  Instituto de Computação – Universidade Estadual de Campinas
- Profa. Dra. Maria Emilia Machado Telles Walter
  Instituto de Ciências Exatas – Universidade de Brasília
- Profa. Dra. Cristina Gomes Fernandes
  Instituto de Matemática e Estatística – Universidade de São Paulo
- Prof. Dr. Eduardo Candido Xavier
  Instituto de Computação – Unicamp
- Prof. Dr. Fábio Luiz Usberti
  Instituto de Computação – Unicamp

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Campinas, 15 de dezembro de 2016

Flipping Hell

Order from disorder.
The pancake stack's in disarray.
How many flips to arrange the stack
back to an ordered way?
Jacob Goodman
(aka Harry Dweighter)
proposed the problem,
(so easy to state),
back in nineteen seventy-five.
But forty years later
it's still very much alive and kicking,
it hasn't been solved,
it needs some licking.

Bill Gates resolved to have a go
and showed with absolute certainty
an upper limit of '5n plus 5, all over three'.
But the problem persists,
no formula exists,
it resists the best minds in the game.
So if you're seeking fame,
want to make your name,
make it next year's resolution,
flipping pancakes 'til you find a solution.

(Unknown author[a])

[a] Found at pherecrates1.wordpress.com.

Agradecimentos

Em 30 de novembro de 2011 recebi um e-mail da Comissão de Pós-Graduação do Instituto de Computação da Unicamp com o título "Parabéns! Você foi aceito no Mestrado do IC/UNICAMP". No início do primeiro semestre de 2012, portanto, saí da minha querida cidade, fui para longe de tudo e todos que conhecia, e hoje, quatro anos depois, inúmeros acontecimentos depois, saio daqui com o título de Doutora.

Aliás, meu pai, Pedrinho, sempre quis ter uma filha "doutora" e, de certa forma, estou realizando esse sonho para ele. Ele pôde me visitar poucas vezes em Campinas, infelizmente, mas jamais (e eu quero dizer jamais mesmo) eu teria chegado até aqui sem ele. Sem todos dias e noites que ele pode trabalhar para que eu pudesse simplesmente estudar e chegar onde cheguei. Foi também com ele que aprendi a me dedicar ao meu trabalho. Obrigada, pai, por tudo!

Minha mãe, Marli, por outro lado, me visitou algumas vezes mais do que meu pai. Foi em Campinas que pude realizar um sonho antigo que ela tinha de andar de trem novamente. O que é muito pouco, considerando que graças a ela eu sempre pude chegar em casa e trabalhar um pouco mais, sem ter outras preocupações. Sair de casa não foi fácil! Com ela aprendi a vida. Obrigada, mãe!

Pelo menos eu não saí de casa sozinha. Meu marido, Maycon, também veio na bagagem. Definitivamente, sem ele eu não teria conseguido obter esse título, mesmo porque foi dele a ideia inicial que devíamos fazer pós-graduação. Obrigada por me aceitar como sou, me ajudar, me aturar, e me completar.

E quando finalmente cheguei ao IC, tive a sorte de ser atribuída a um tutor muito dedicado, Zanoni, que eventualmente se tornou meu orientador e amigo. Não tenho meios de agradecer a confiança que ele depositou em mim desde o início e as inúmeras vezes em que me ajudou, a crescer profissionalmente mas também de outras várias formas.

A mudança para Campinas foi um divisor de águas na minha vida. Cresci bastante, viajei algumas vezes, conheci pessoas e tive novas experiências. Na mais importante destas morei por seis meses sozinha em Nantes, França, onde tive a oportunidade de trabalhar com Guillaume Fertin, ao qual tenho também muito a agradecer por toda dedicação.

Agradeço também à Fapesp, pelo essencial apoio financeiro concedido para a realização deste doutorado tanto aqui no Brasil quanto na França (processos 2013/01172-0 e 2014/20738-7) e ao CNPq, pelo apoio financeiro inicial (processo 140017/2013-5). Agradeço ainda ao LOCo e ao CCES CEPID/FAPESP pelo recursos computacionais.

Muito do que sou hoje também é reflexo de quem conheci. Meus familiares e os amigos que guardo até hoje são pessoas boas que fizeram a diferença. Não são muitos, mas citar todos pode ser perigoso devido à minha leve falta de memória (e ao máximo de uma página desta seção). Agradeço a todos de coração.

No final das contas, tenho muito a agradecer a Deus por todas as oportunidades que tive e, principalmente, por todas essas pessoas que fazem parte da minha vida. Espero ainda poder acrescentar mais pessoas e agradecimentos nessa história.

Resumo

O Problema das Panquecas tem como objetivo ordenar uma pilha de panquecas que possuem tamanhos distintos realizando o menor número possível de operações. A operação permitida é chamada reversão de prefixo e, quando aplicada, inverte o topo da pilha de panquecas. Tal problema é interessante do ponto de vista combinatório por si só, mas ele também possui algumas aplicações em biologia computacional. Dados dois genomas que compartilham o mesmo número de genes, e assumindo que cada gene aparece apenas uma vez por genoma, podemos representá-los como permutações (pilhas de panquecas também são representadas por permutações). Então, podemos comparar os genomas tentando descobrir como um foi transformado no outro por meio da aplicação de rearranjos de genoma, que são eventos de mutação de grande escala. Reversões e transposições são os tipos mais comumente estudados de rearranjo de genomas e uma reversão de prefixo (ou transposição de prefixo) é um tipo de reversão (ou transposição) que é restrita ao início da permutação. Quando o rearranjo é restrito ao final da permutação, dizemos que ele é um rearranjo de sufixo.

Um problema de ordenação de permutações por rearranjos é, portanto, o problema de encontrar uma sequência de rearranjos de custo mínimo que ordene a permutação dada. A abordagem *tradicional* considera que todos os rearranjos têm o mesmo custo unitário, de forma que o objetivo é tentar encontrar o menor número de rearranjos necessários para ordenar a permutação. Vários esforços foram feitos nos últimos anos considerando essa abordagem. Por outro lado, um rearranjo muito longo (que na verdade é uma mutação) tem mais probabilidade de perturbar o organismo. Portanto, pesos baseados no comprimento do segmento envolvido podem ter um papel importante no processo evolutivo. Dizemos que essa abordagem é *ponderada por comprimento* e o objetivo nela é tentar encontrar uma sequência de rearranjos cujo custo total (que é a soma do custo de cada rearranjo, que por sua vez depende de seu comprimento) seja mínimo.

Nessa tese nós apresentamos os primeiros resultados que envolvem problemas de ordenação de permutações por reversões e transposições de prefixo e sufixo considerando ambas abordagens tradicional e ponderada por comprimento. Na abordagem tradicional, consideramos um total de 10 problemas e desenvolvemos novos resultados para 6 deles. Na abordagem ponderada por comprimento, consideramos um total de 13 problemas e desenvolvemos novos resultados para todos eles.

Abstract

The goal of the Pancake Flipping problem is to sort a stack of pancakes that have different sizes by performing as few operations as possible. The operation allowed is called prefix reversal and, when applied, flips the top of the stack of pancakes. Such problem is an interesting combinatorial problem by itself, but it has some applications in computational biology. Given two genomes that share the same genes and assuming that each gene appears only once per genome, we can represent them as permutations (stacks of pancakes are also represented by permutations). Then, we can compare the genomes by figuring out how one was transformed into the other through the application of genome rearrangements, which are large scale mutations. Reversals and transpositions are the most commonly studied types of genome rearrangements and a prefix reversal (or prefix transposition) is a type of reversal (or transposition) which is restricted to the beginning of the permutation. When the rearrangement is restricted to the end of the permutation, we say it is a suffix rearrangement.

A problem of sorting permutations by rearrangements is, therefore, the problem to find a sequence of rearrangements with minimum cost that sorts a given permutation. The *traditional* approach considers that all rearrangements have the same unitary cost, in which case the goal is trying to find the minimum number of rearrangements that are needed to sort the permutation. Numerous efforts have been made over the past years regarding this approach. On the other hand, a long rearrangement (which is in fact a mutation) is more likely to disturb the organism. Therefore, weights based on the length of the segment involved may have an important role in the evolutionary process. We say this is the *length-weighted* approach and the goal is trying to find a sequence of rearrangements whose total cost (the sum of the cost of each rearrangement, which depends on its length) is minimum.

In this thesis we present the first results regarding problems of sorting permutations by prefix and suffix reversals and transpositions considering both the traditional and the length-weighted approach. For the traditional approach, we considered a total of 10 problems and developed new results for 6 of them. For the length-weighted approach, we considered a total of 13 problems and developed new results for all of them.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

One of the challenges of modern science is to understand how evolution happened, considering that new organisms arise from mutations that occurred in others: different species may share some similar genes that are not necessarily in the same order or even orientation when they are compared. The evolutionary distance between two organisms may be inferred through the genome rearrangement distance between them, which considers a sequence of genome rearrangement events that occurred in the transformation of the genome of one organism into the genome of another. A *genome rearrangement* is a type of large scale *mutation* that can occur in a genome. A *reversal*, for instance, is a genome rearrangement that inverts a segment of a genome. Another well-known rearrangement is the *transposition*, which exchanges the position of two consecutive segments of a genome.

Depending on the assumptions that are made or the problems considered, different models of genomes can be used. The simplest one considers that the genomes contain a single copy of each gene and that all genomes consist of a single chromosome. This allows us to model them as *permutations*. The *unsigned* variant considers that the orientation of the genes is not known while the *signed* variant indicates it with '+' or '-' signs on each element of the permutation. The goal then is to find a sequence of genome rearrangements that occurred over one permutation so that it could be turned into the other. The use of permutations allows us to consider that one of them is the identity so that we only need to sort the other. We then call such problems of *sorting by rearrangements*.

For such problems, it is common to find in the literature two types of approaches. The first one considers a given permutation and wants to find its *distance*, which is the minimum cost of a sequence of rearrangements that sorts it. The second one considers all permutations of a given size and wants to find the greatest distance between all of them, which is called the *diameter*.

The study of problems of sorting permutations by rearrangements started being interesting by itself, in such a way that some new variations of genome rearrangements started to be considered, such as *prefix reversals* and *prefix transpositions*, which always involve the first element of the permutation. The famous Pancake Flipping problem, for instance, was introduced in 1975 by Harry Dweighter (pseudonym of Jacob Eli Goodman) [25] with the story of a waiter that receives a stack of pancakes with different sizes and needs to sort it in such a way that the smallest pancake ends up at the top, the second smallest ends up below the smallest, and so on, until the largest pancake ends up at the bottom

of the stack. To do this sorting, the waiter can only flip a block of pancakes from the top of the stack. He wants to know what is the maximum number of flips that he will ever have to use to sort any stack of n pancakes. In other words, it describes the problem of sorting permutations by prefix reversals. Even so, these variations can help us with some insights into the non-restricted variations.

During the past years, there were several studies considering sorting permutations by reversals [2, 7, 11], by signed reversals [1, 37, 55], by transpositions [9, 26, 27], by reversals and transpositions [51, 56], by signed reversals and transpositions [56], by prefix reversals [10, 15, 30, 34, 40], by signed prefix reversals [17, 18], by prefix transpositions [14, 24, 42], and by prefix reversals and transpositions [23, 53]. We say these belong to the *traditional approach* of the sorting problems, where the distance of a permutation is the minimum amount of rearrangements needed to sort it.

On the other hand, not so many studies consider the *length-weighted approach*, where the cost function to calculate the distance depends on the length of the rearrangements [4, 5, 20, 49, 50, 54]. Such studies recognize that a very long rearrangement, which is in fact a mutation, is more likely to disturb the organism. This is modeled by a cost function which increases as the length of the rearrangement increases. Furthermore, some preliminary results show that in some cases the reversals that happened during evolution indeed tend not to be very long [8], which further indicates that weights based on the length of the segment involved may have an important role in the evolutionary process. We call such problems as *sorting by length-weighted rearrangements*.

During this doctorate, we mainly studied variants of rearrangements that are restricted to the prefix and the suffix of a permutation, that is, variations of genome rearrangements that always involve the first or the last element. This thesis describes our results for both the traditional approach and the length-weighted approach. As it was shown by Bender *et al.* [5], binary strings, which are strings that contain only the symbols 0 and 1, are closely related to sorting permutations by length-weighted rearrangements. For this reason, we also show some results for sorting binary strings by length-weighted rearrangements. The names and acronyms of all the problems that we somehow considered are listed in Table 1.1.

We note that the use of more than one type of rearrangement while doing the sorting allows some reduction on the distance of the permutations, as one can see in Table 1.2, which shows, for small values of n, the percentage of permutations for which the distance when using prefix and suffix rearrangements is strictly smaller than the distance when using only prefix rearrangements (considering the traditional approach). The number in parenthesis in this table is the maximum difference between the prefix and suffix distance and the prefix distance. We note that Fertin *et al.* [29] showed that, for infinitely many values of n, there is a permutation of size n such that the difference between the prefix reversal distance and the prefix and suffix reversal distance is $\Omega(n)$.

The rest of this thesis is divided as follows. Chapter 2 gives important definitions that are used throughout the document. Chapters 3 and 4 consider the traditional approach, that is, for which the cost of sorting a permutation is given by the amount of rearrangements that were used to do it. The former gives a review of the literature while the latter presents the results we obtained. Chapters 5 and 6 consider the length-weighted

Table 1.1: Acronyms of the problems mentioned and studied on this thesis.

Sorting by Reversals	SbR
Sorting by Signed Reversals	SbR̄
Sorting by Transpositions	SbT
Sorting by Reversals and Transpositions	SbRT
Sorting by Signed Reversals and Transpositions	SbR̄T
Sorting by Prefix Reversals	SbPR
Sorting by Signed Prefix Reversals	SbPR̄
Sorting by Prefix Transpositions	SbPT
Sorting by Prefix Reversals and Transpositions	SbPRT
Sorting by Signed Prefix Reversals and Transpositions	SbPR̄T
Sorting by Prefix and Suffix Reversals	SbPSR
Sorting by Signed Prefix and Suffix Reversals	SbPSR̄
Sorting by Prefix and Suffix Transpositions	SbPST
Sorting by Prefix and Suffix Reversals and Transpositions	SbPSRT
Sorting by Signed Prefix and Suffix Reversals and Transpositions	SbPSR̄T
Sorting by Length-Weighted Reversals	SbWR
Sorting by Length-Weighted Signed Reversals	SbWR̄
Sorting by Length-Weighted Transpositions	SbWT
Sorting by Length-Weighted Reversals and Transpositions	SbWRT
Sorting by Length-Weighted Signed Reversals and Transpositions	SbWR̄T
Sorting by Length-Weighted Prefix Reversals	SbWPR
Sorting by Length-Weighted Signed Prefix Reversals	SbWPR̄
Sorting by Length-Weighted Prefix Transpositions	SbWPT
Sorting by Length-Weighted Prefix Reversals and Transpositions	SbWPRT
Sorting by Length-Weighted Signed Prefix Reversals and Transpositions	SbWPR̄T
Sorting by Length-Weighted Prefix and Suffix Reversals	SbWPSR
Sorting by Length-Weighted Signed Prefix and Suffix Reversals	SbWPSR̄
Sorting by Length-Weighted Prefix and Suffix Transpositions	SbWPST
Sorting by Length-Weighted Prefix and Suffix Reversals and Transpositions	SbWPSRT
Sorting by Length-Weighted Signed Prefix and Suffix Reversals and Transpositions	SbWPSR̄T
Sorting Binary Strings by Length-Weighted Reversals	SBbWR
Sorting Binary Strings by Length-Weighted Signed Reversals	SBbWR̄
Sorting Binary Strings by Length-Weighted Transpositions	SBbWT
Sorting Binary Strings by Length-Weighted Reversals and Transpositions	SBbWRT
Sorting Binary Strings by Length-Weighted Signed Reversals and Transpositions	SBbWR̄T
Sorting Binary Strings by Length-Weighted Prefix Reversals	SBbWPR
Sorting Binary Strings by Length-Weighted Signed Prefix Reversals	SBbWPR̄
Sorting Binary Strings by Length-Weighted Prefix Transpositions	SBbWPT
Sorting Binary Strings by Length-Weighted Prefix Reversals and Transpositions	SBbWPRT
Sorting Binary Strings by Length-Weighted Signed Prefix Reversals and Transpositions	SBbWPR̄T
Sorting Binary Strings by Length-Weighted Prefix and Suffix Reversals	SBbWPSR
Sorting Binary Strings by Length-Weighted Signed Prefix and Suffix Reversals	SBbWPSR̄
Sorting Binary Strings by Length-Weighted Prefix and Suffix Transpositions	SBbWPST
Sorting Binary Strings by Length-Weighted Prefix and Suffix Reversals and Transpositions	SBbWPSRT
Sorting Binary Strings by Length-Weighted Signed Prefix and Suffix Reversals and Transpositions	SBbWPSR̄T

Table 1.2: Percentage of permutations for which the distance using prefix and suffix rearrangements is smaller than the distance using prefix rearrangements only (considering the traditional approach). In parenthesis we show the maximum difference between the two distances. Cells with "-" have unknown values.

n	SbPR vs. SbPSR	SbPT vs. SbPST	SbPRT vs. SbPSRT	SbPR vs. SbPS$\bar{\text{R}}$	SbPRT vs. SbPS$\bar{\text{R}}$T
2	0.000% (0)	0.000% (0)	0.000% (0)	25.000% (2)	12.500% (2)
3	16.666% (2)	16.666% (1)	16.666% (1)	54.166% (2)	39.583% (2)
4	54.166% (2)	20.833% (1)	16.666% (1)	68.229% (4)	47.135% (2)
5	60.000% (2)	25.833% (1)	25.833% (2)	76.771% (4)	49.635% (2)
6	68.055% (3)	31.805% (1)	27.222% (2)	82.613% (6)	54.028% (2)
7	74.722% (4)	35.675% (2)	34.583% (2)	86.944% (6)	61.198% (2)
8	80.484% (4)	40.536% (2)	36.889% (2)	90.233% (7)	63.877% (3)
9	85.023% (4)	44.381% (2)	41.617% (2)	92.726% (7)	66.736% (3)
10	88.378% (4)	47.901% (2)	45.416% (2)	-	-

approach, for which the cost of sorting a permutation is calculated based on the length of the rearrangements involved in the process. As before, the former gives a review of the literature while the latter presents the results we obtained. Chapter 7 presents results that we obtained when still considering the length of the rearrangements involved in the sorting process, but with another cost function. Chapter 8 gives some final considerations and summarizes our contributions.

Chapter 2

Theoretical Fundaments

This chapter gives some important definitions that will be used throughout this thesis in more than one chapter. Some local definitions may still be given in future chapters, whenever necessary.

Permutations are one of the mathematical models used to formalize the study of DNA fragments [28]. They represent the chromosomes as sequences of segments that are being shared by the genomes we are comparing, and if the orientation of the genes is known, a signed permutation is normally used.

Definition 1. *An* unsigned permutation *of size n is represented as $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$, where $\pi_i \in \{1, 2, \ldots, n\}$ for all i such that $1 \leq i \leq n$ and $\pi_i \neq \pi_j$ whenever $i \neq j$.*

Definition 2. *A* signed permutation *of size n is represented as $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$, where $\pi_i \in \{-n, -(n-1), \ldots, -1, 1, 2, \ldots, n\}$ for all i such that $1 \leq i \leq n$ and $|\pi_i| \neq |\pi_j|$ whenever $i \neq j$.*

In order to simplify some computations, we always consider the *extended* version of a permutation, in which there are elements $\pi_0 = 0$ and $\pi_{n+1} = n + 1$ that are fixed, even if we do not show them explicitly.

There exist some special permutations that may be of importance in some occasions. The *identity* permutation $\iota_n = (1\ 2\ 3\ \ldots\ n)$, for instance, always represents the goal of the sorting problems we will study. We also have the *reverse* permutation, $\eta_n = (n\ n-1\ n-2\ \ldots\ 1)$, and the *signed reverse* permutation, $\bar{\eta_n} = (-n\ -(n-1)\ -(n-2)\ \ldots\ -1)$. The *inverse* permutation of π is π^{-1}, for which $\pi^{-1}_{|\pi_i|} = i$, where $1 \leq i \leq n$. We at last define $\varphi^a_{k,n} = (k\ k+1\ k+2\ \ldots\ n\ 1\ 2\ 3\ \ldots\ k-1)$ (where the a stands for ascending), $\varphi^d_{k,n} = (k-1\ k-2\ k-3\ \ldots\ 1\ n\ n-1\ n-2\ \ldots\ k)$ (where the d stands for descending), and $\varphi^s_{k,n} = (-(k-1)\ -(k-2)\ -(k-3)\ \ldots\ -1\ -n\ -(n-1)\ -(n-2)\ \ldots\ -k)$ (where the s stands for signed), for any k such that $2 \leq k \leq n$.

Definition 3. *A* composition *of two permutations π and σ is the operation denoted by "$\cdot$", for which $\pi \cdot \sigma = (\pi_{\sigma_1}\ \pi_{\sigma_2}\ \ldots\ \pi_{\sigma_n})$. Note that $\pi \cdot \pi^{-1} = \pi^{-1} \cdot \pi = \iota_n$.*

We also use permutations to represent the rearrangements so that a certain rearrangement λ transforms a permutation π into another permutation $\pi \cdot \lambda$. Therefore, the composition is used to indicate the occurrence of a rearrangement on a permutation.

Definition 4. *A* reversal $\rho(i, j)$*, for* $1 \leq i < j \leq n$*, is the rearrangement* $(1\ 2\ \ldots\ i{-}1\ \underline{j\ j{-}1\ \ldots\ i{+}1\ i}\ j{+}1\ j{+}2\ \ldots\ n)$*, which means that, when applied to a permutation* π*, it inverts the segment that goes from position* i *to position* j*, transforming* π *into* $\pi \cdot \rho(i, j) = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{\pi_j\ \pi_{j-1}\ \ldots\ \pi_{i+1}\ \pi_i}\ \pi_{j+1}\ \ldots\ \pi_n)$*. We define a* prefix reversal $\rho_p(j)$ *as the reversal* $\rho(1, j)$*,* $1 < j \leq n$*, and a* suffix reversal $\rho_s(i)$ *as the reversal* $\rho(i, n)$*,* $1 \leq i < n$*.*

Definition 5. *A* signed reversal $\bar{\rho}(i, j)$*, for* $1 \leq i \leq j \leq n$*, is the rearrangement* $(1\ 2\ \ldots\ i{-}1\ \underline{{-}j\ {-}(j{-}1)\ \ldots\ {-}(i{+}1)\ {-}i}\ j{+}1\ j{+}2\ \ldots\ n)$*, which means that, when applied to a signed permutation* π*, it inverts the segment that goes from position* i *to position* j *and it changes the signs of each element of the segment, transforming* π *into* $\pi \cdot \bar{\rho}(i, j) = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{{-}\pi_j\ {-}\pi_{j-1}\ \ldots\ {-}\pi_{i+1}\ {-}\pi_i}\ \pi_{j+1}\ \ldots\ \pi_n)$*. We define a* signed prefix reversal $\bar{\rho}_p(j)$ *as the signed reversal* $\bar{\rho}(1, j)$*,* $1 \leq j \leq n$*, and a* signed suffix reversal $\bar{\rho}_s(i)$ *as the signed reversal* $\bar{\rho}(i, n)$*,* $1 \leq i \leq n$*.*

Definition 6. *A* transposition $\tau(i, j, k)$*, for* $1 \leq i < j < k \leq n + 1$*, is the rearrangement* $(1\ 2\ \ldots\ i{-}1\ \underline{j\ j{+}1\ \ldots\ k{-}1}\ \underline{i\ i{+}1\ \ldots\ j{-}1}\ k\ k{+}1\ \ldots\ n)$*, which means that, when applied to a permutation* π*, it exchanges the segment that goes from position* i *to* $j{-}1$ *with the segment that goes from position* j *to* $k{-}1$*, transforming* π *into* $\pi \cdot \tau(i, j, k) = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{\pi_j\ \pi_{j-1}\ \ldots\ \pi_{k-1}}\ \underline{\pi_i\ \pi_{i+1}\ \ldots\ \pi_{j-1}}\ \pi_k\ \ldots\ \pi_n)$*. We define a* prefix transposition $\tau_p(j, k)$ *as the transposition* $\tau(1, j, k)$*,* $1 < j < k \leq n + 1$*, and a* suffix transposition $\tau_s(i, j)$ *as the transposition* $\tau(i, j, n + 1)$*,* $1 \leq i < j < n + 1$*.*

We say that *general rearrangements* are those which are not constrained to be prefix or suffix.

Example 1. *Let* $\pi = (3\ 8\ 4\ 1\ 2\ 5\ 9\ 6\ 10\ 7)$*. We have* $\pi \cdot \rho(3, 7) = (3\ 8\ 9\ 5\ 2\ 1\ 4\ 6\ 10\ 7)$*,* $\pi \cdot \bar{\rho}(3, 7) = (3\ 8\ {-}9\ {-}5\ {-}2\ {-}1\ {-}4\ 6\ 10\ 7)$*, and* $\pi \cdot \tau(2, 6, 8) = (3\ 5\ 9\ 8\ 4\ 1\ 2\ 6\ 10\ 7)$*.*

Note that there is not a signed version for transpositions, since this rearrangement only exchanges the position of elements. From now on, we will make it clear only when we are dealing with signed permutations or rearrangements (the "unsigned" terms will be omitted).

A *rearrangement model* β is the set of rearrangements that are allowed during the sorting process. In order to simplify, we do not use the usual set notation; instead, we will write β as "$(\ |p|ps)(r|\bar{r}|t|rt|\bar{r}t)$" where p stands for prefix, ps for prefix and suffix, r for reversals, $\bar{r}$ for signed reversals, t for transpositions, rt for reversals and transpositions, and $\bar{r}t$ for signed reversals and transpositions. For instance, if $\beta = rt$, then reversals and transpositions are allowed, or if $\beta = psrt$, then prefix reversals, prefix transpositions, suffix reversals, and suffix transpositions are allowed.

Definition 7. *The* length ℓ *of a reversal* $\rho(i, j)$ *is defined as* $j - i + 1$ *while the length* ℓ *of a transposition* $\tau(i, j, k)$ *is defined as* $k - i$*. We define* $f(\ell) = \ell^\alpha$ *as the* cost *of a rearrangement that has length* ℓ*, for* $\alpha \geq 0$*.*

We say that a sequence of rearrangements is a *sorting sequence* for a permutation π if they transform π into ι_n when they are applied to π by composition. Consider a sorting sequence for π of k rearrangements such that the lengths of these rearrangements are ℓ_1, ℓ_2, ..., ℓ_k. We define the *cost* of such sequence as $f(\ell_1) + f(\ell_2) + \cdots + f(\ell_k)$.

Example 2. *Let $\beta = pr$ and $\pi = (3\ 7\ 5\ 1\ 4\ 2\ 6)$. A possible sorting sequence for π is shown next (each prefix reversal is underlined):*

$$\begin{aligned}
\pi &= (\underline{3\ 7}\ 5\ 1\ 4\ 2\ 6)\\
\pi{\leftarrow}\pi\cdot\rho_p(2) &= (\underline{7\ 3\ 5}\ 1\ 4\ 2\ 6)\\
\pi{\leftarrow}\pi\cdot\rho_p(3) &= (\underline{5\ 3\ 7\ 1}\ 4\ 2\ 6)\\
\pi{\leftarrow}\pi\cdot\rho_p(4) &= (\underline{1\ 7}\ 3\ 5\ 4\ 2\ 6)\\
\pi{\leftarrow}\pi\cdot\rho_p(2) &= (\underline{7\ 1\ 3\ 5\ 4\ 2\ 6})\\
\pi{\leftarrow}\pi\cdot\rho_p(7) &= (\underline{6\ 2\ 4}\ 5\ 3\ 1\ 7)\\
\pi{\leftarrow}\pi\cdot\rho_p(3) &= (\underline{4\ 2}\ 6\ 5\ 3\ 1\ 7)\\
\pi{\leftarrow}\pi\cdot\rho_p(2) &= (\underline{2\ 4\ 6\ 5}\ 3\ 1\ 7)\\
\pi{\leftarrow}\pi\cdot\rho_p(4) &= (\underline{5\ 6}\ 4\ 2\ 3\ 1\ 7)\\
\pi{\leftarrow}\pi\cdot\rho_p(2) &= (\underline{6\ 5\ 4\ 2\ 3\ 1}\ 7)\\
\pi{\leftarrow}\pi\cdot\rho_p(6) &= (\underline{1\ 3}\ 2\ 4\ 5\ 6\ 7)\\
\pi{\leftarrow}\pi\cdot\rho_p(2) &= (\underline{3\ 1\ 2}\ 4\ 5\ 6\ 7)\\
\pi{\leftarrow}\pi\cdot\rho_p(3) &= (\underline{2\ 1}\ 3\ 4\ 5\ 6\ 7)\\
\pi{\leftarrow}\pi\cdot\rho_p(2) &= (1\ 2\ 3\ 4\ 5\ 6\ 7)
\end{aligned}$$

For $\alpha = 0$, the cost of this sorting sequence is 13 (there are 13 prefix reversals in it). For $\alpha = 1$, the cost is 42 (because $2 + 3 + 4 + 2 + 7 + 3 + 2 + 4 + 2 + 6 + 2 + 3 + 2 = 42$). And when $\alpha = 5$, the cost is $27,552$ (because $2^5 + 3^5 + 4^5 + 2^5 + 7^5 + 3^5 + 2^5 + 4^5 + 2^5 + 6^5 + 2^5 + 3^5 + 2^5 = 27,552$).

The value of $f(\ell)$ has usually been considered as the polynomial function ℓ^α [5, 49, 54]. Below we give some properties of $f(x) = x^\alpha$, $\alpha \geq 0$, which will be useful during Chapter 6:

1. Unitary cost ($\alpha = 0$): $f(x) = 1$;
2. Sub-additive cost ($\alpha < 1$): $f(x) + f(y) > f(x + y)$;
3. Additive cost ($\alpha = 1$): $f(x) + f(y) = f(x + y)$;
4. Super-additive cost ($\alpha > 1$): $f(x) + f(y) < f(x + y)$.

In Chapter 7 we present some results considering an exponential cost function, namely $f(\ell) = 2^\ell$. Until there, we will always consider $f(\ell) = \ell^\alpha$.

These definitions of cost finally allow us to define the goal of all the problems we will deal with.

Definition 8. *Consider a rearrangement model β and some $\alpha \geq 0$. The* distance *of a permutation π is defined as the cost of a sorting sequence for π of minimum cost and it is denoted as $d^\alpha_\beta(\pi)$.*

Definition 9. *A* problem of sorting permutations by a rearrangement model β *receives as input one permutation π (signed or unsigned) and a real $\alpha \geq 0$, and consists of finding the minimum cost to transform π into the identity permutation. In other words, it consists of finding the distance $d^\alpha_\beta(\pi)$.*

Example 3. *Let $\beta = pr$ and $\pi = (3\ 7\ 5\ 1\ 4\ 2\ 6)$, as in Example 2. The sorting sequence given there has the minimum cost neither when $\alpha = 0$ nor when $\alpha = 1$, for example. However, it allows us to say that $d^0_{pr}(\pi) \leq 13$ and $d^1_{pr}(\pi) \leq 42$. On the other hand, that sequence does have the minimum cost when $\alpha = 5$. Therefore, $d^5_{pr}(\pi) = 27,552$. Next we*

give two minimum sorting sequences for $\alpha = 0$ and $\alpha = 1$:

$$\begin{aligned}
\pi &= (\underline{3\ 7\ 5\ 1\ 4}\ 2\ 6)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(5) &= (\underline{4\ 1\ 5\ 7}\ 3\ 2\ 6)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(4) &= (\underline{7\ 5\ 1\ 4\ 3\ 2\ 6})\\
\pi{\leftarrow}\pi{\cdot}\rho_p(7) &= (\underline{6\ 2\ 3\ 4\ 1\ 5}\ 7)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(6) &= (\underline{5\ 1\ 4\ 3\ 2}\ 6\ 7)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(5) &= (\underline{2\ 3\ 4}\ 1\ 5\ 6\ 7)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(3) &= (\underline{4\ 3\ 2\ 1}\ 5\ 6\ 7)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(4) &= (1\ 2\ 3\ 4\ 5\ 6\ 7)
\end{aligned}$$

$$\begin{aligned}
\pi &= (\underline{3\ 7}\ 5\ 1\ 4\ 2\ 6)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(2) &= (\underline{7\ 3\ 5\ 1\ 4\ 2\ 6})\\
\pi{\leftarrow}\pi{\cdot}\rho_p(7) &= (\underline{6\ 2\ 4\ 1\ 5\ 3}\ 7)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(6) &= (\underline{3\ 5}\ 1\ 4\ 2\ 6\ 7)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(2) &= (\underline{5\ 3\ 1\ 4\ 2}\ 6\ 7)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(5) &= (\underline{2\ 4}\ 1\ 3\ 5\ 6\ 7)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(2) &= (\underline{4\ 2\ 1\ 3}\ 5\ 6\ 7)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(4) &= (\underline{3\ 1\ 2}\ 4\ 5\ 6\ 7)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(3) &= (\underline{2\ 1}\ 3\ 4\ 5\ 6\ 7)\\
\pi{\leftarrow}\pi{\cdot}\rho_p(2) &= (1\ 2\ 3\ 4\ 5\ 6\ 7)
\end{aligned}$$

For $\alpha = 0$, the sequence at the left costs 7 and there does not exist a sequence with less than 7 prefix reversals that sorts π ($d^0_{pr}(\pi) = 7$). Note that this sequence has a cost of 34 when $\alpha = 1$, but the distance in this case is 33 ($d^1_{pr}(\pi) = 33$) and the sequence at the right has such cost.

Note that computing $d^\alpha_\beta(\pi)$ for all permutations of a certain size n is not trivial, specially as n grows. Galvão and Dias [31], for instance, presented an algorithm such that, given a rearrangement model and considering $\alpha = 0$, calculates the distance and shows the sorting sequence of any permutation of a given size n. Considering the memory limitation of the machines used, it currently has information for all unsigned permutations with $n \leq 13$ and for all signed permutations with $n \leq 11$.

Definition 10. *Consider a rearrangement model β and some $\alpha \geq 0$. The* diameter *of a problem of sorting permutations is the largest distance among the distances of all permutations of size n and it is denoted as $D^\alpha_\beta(n)$. Specifically, $D^\alpha_\beta(n) = \max\{d^\alpha_\beta(\pi) : \pi$ has size $n\}$.*

As secondary goals, given a problem of sorting by genome rearrangements, we can also study the diameters. Also note that computing $D^\alpha_\beta(n)$ is not trivial, specially if we note that there are $n!$ unsigned permutations of size n and $2^n n!$ signed permutations of size n. Such studies are of importance, for instance, in network design [40]. A pancake network has $n!$ processors, each one labeled with one of the possible permutations of size n, and there is a link between two processors if the label of one of them is obtained through a prefix reversal from the label of the other. The diameter of this network is the maximum distance between any two processors while the distance between two given processor is the length of the minimum path between both.

Example 4. *Let $n = 4$, $\beta = pr$, and $\alpha = 0$. For the 16 possible permutations of size 4, we have that $d^0_{pr}(1\ 2\ 3\ 4) = 0$, $d^0_{pr}(1\ 2\ 4\ 3) = 3$, $d^0_{pr}(1\ 3\ 2\ 4) = 3$, $d^0_{pr}(1\ 3\ 4\ 2) = 3$, $d^0_{pr}(1\ 4\ 2\ 3) = 3$, $d^0_{pr}(1\ 4\ 3\ 2) = 3$, $d^0_{pr}(2\ 1\ 3\ 4) = 1$, $d^0_{pr}(2\ 1\ 4\ 3) = 3$, $d^0_{pr}(2\ 3\ 1\ 4) = 2$, $d^0_{pr}(2\ 3\ 4\ 1) = 2$, $d^0_{pr}(2\ 4\ 1\ 3) = 4$, $d^0_{pr}(2\ 4\ 3\ 1) = 3$, $d^0_{pr}(3\ 1\ 2\ 4) = 2$, $d^0_{pr}(3\ 1\ 4\ 2) = 4$, $d^0_{pr}(3\ 2\ 1\ 4) = 1$, $d^0_{pr}(3\ 2\ 4\ 1) = 3$, $d^0_{pr}(3\ 4\ 1\ 2) = 3$, $d^0_{pr}(3\ 4\ 2\ 1) = 2$, $d^0_{pr}(4\ 1\ 2\ 3) = 2$, $d^0_{pr}(4\ 1\ 3\ 2) = 3$, $d^0_{pr}(4\ 2\ 1\ 3) = 3$, $d^0_{pr}(4\ 2\ 3\ 1) = 4$, $d^0_{pr}(4\ 3\ 1\ 2) = 2$, and $d^0_{pr}(4\ 3\ 2\ 1) = 1$. Since the maximum distance achieved is 4, the diameter is 4 ($D^0_{pr}(4) = 4$).*

Historically, the distance of a permutation when $\alpha = 0$ (which means that $f(\ell) = 1$) is denoted as $d_\beta(\pi)$ and the diameter for n is denoted as $D_\beta(n)$. We will maintain such conventions by defining $d_\beta(\pi) = d^0_\beta(\pi)$ and $D_\beta(n) = D^0_\beta(n)$. Note that the distance in this traditional approach is simply the minimum number of rearrangements needed to sort a given permutation.

2.1 Breakpoints and Strips

The concept of *breakpoints* is very common for genome rearrangement problems. For this thesis, we need to define eight types of breakpoints, being four of the unsigned types and other four of the signed/transposition types. In general, breakpoints are pairs of consecutive elements that should not be consecutive at the end of the sorting, that is, in the identity permutation. If a pair of consecutive elements is not a breakpoint, then we say that it is an *adjacency*.

Definition 11. *For the unsigned types, the general idea is that a breakpoint is a pair* (π_i, π_{i+1}) *of consecutive elements such that* $|\pi_{i+1} - \pi_i| \neq 1$. *Index* i *varies according to the problem that we are dealing with:*

- *For* SbR, SbRT, SbWR, *and* SbWRT *we call them* general reversal breakpoints, *or* gr-breakpoints, *and* $0 \leq i \leq n$;
- *For* SbPR, SbPRT, SbWPR, *and* SbWPRT *we call them* unsigned prefix reversal breakpoints, *or* upr-breakpoints, *and* $1 \leq i \leq n$. *Note that* (π_0, π_1) *is, by definition, not a upr-breakpoint;*
- *For* SbPSR *and* SbWPSR, *we call them* unsigned prefix and suffix reversal breakpoints, *or* upsr-breakpoints, *and* $1 \leq i < n$. *Note that, by definition, neither* (π_0, π_1) *nor* (π_n, π_{n+1}) *are upsr-breakpoints;*
- *For* SbPSRT *and* SbWPSRT, *we call them* unsigned prefix and suffix reversal and transposition breakpoints, *or upsrt-breakpoints, and* $1 \leq i < n$. *Note that, by definition, neither* (π_0, π_1) *nor* (π_n, π_{n+1}) *are upsrt-breakpoints. We also impose, by definition, that* $(1, n)$ *and* $(n, 1)$ *are never upsrt-breakpoints.*

We denote the number of gr-breakpoints, upr-breakpoints, upsr-breakpoints, and upsrt-breakpoints in a permutation π as $b_{gr}(\pi)$, $b_{upr}(\pi)$, $b_{upsr}(\pi)$, and $b_{upsrt}(\pi)$, respectively. We can easily see that $b_{gr}(\iota_n) = b_{upr}(\iota_n) = b_{upsr}(\iota_n) = b_{upsrt}(\iota_n) = 0$, $b_{upsr}(\eta_n) = b_{upsrt}(\eta_n) = 0$, and $b_{upsrt}(\varphi^a_{k,n}) = b_{upsrt}(\varphi^d_{k,n}) = 0$ for any $2 \leq k \leq n$. Note that these are the only permutations with zero breakpoints in the unsigned types.

Example 5. *Next we show, for each of the four types of breakpoints defined above, how to count them over the same two permutations. A breakpoint is indicated by a "*.*":*

gr-breakpoints:	(1 . 4 3 . 8 . 5 6 . 2 . 7 .)	(. 5 6 . 2 . 8 . 1 . 4 3 . 7 .)
upr-breakpoints:	(1 . 4 3 . 8 . 5 6 . 2 . 7 .)	(5 6 . 2 . 8 . 1 . 4 3 . 7 .)
upsr-breakpoints:	(1 . 4 3 . 8 . 5 6 . 2 . 7)	(5 6 . 2 . 8 . 1 . 4 3 . 7)
upsrt-breakpoints:	(1 . 4 3 . 8 . 5 6 . 2 . 7)	(5 6 . 2 . 8 1 . 4 3 . 7)

Definition 12. *For the signed/transposition types, the general idea is that a breakpoint is a pair* (π_i, π_{i+1}) *of consecutive elements such that* $\pi_{i+1} - \pi_i \neq 1$. *Again, index* i *varies according to the problem that we are dealing with:*

- *For* Sb$\bar{\text{R}}$, SbT, Sb$\bar{\text{R}}$T Sb$\text{W}\bar{\text{R}}$, SbWT, *and* Sb$\text{W}\bar{\text{R}}$T *we call them* general breakpoints, *or* g-breakpoints, *and* $0 \leq i \leq n$;

- *For* SBPT, SBP$\bar{\text{R}}$, SBP$\bar{\text{R}}$T, SBWPT, SBWP$\bar{\text{R}}$, *and* SBWP$\bar{\text{R}}$T *we call them* prefix breakpoints, *or* p-breakpoints, *and* $1 \leq i \leq n$. *Note that* (π_0, π_1) *is never a p-breakpoint;*

- *For* SBPST, SBPS$\bar{\text{R}}$, SBWPST, *and* SBWPS$\bar{\text{R}}$ *we call them* prefix and suffix breakpoints, *or* ps-breakpoints, *and* $1 \leq i < n$. *Note that neither* (π_0, π_1) *nor* (π_n, π_{n+1}) *are ps-breakpoints;*

- *For* SBPS$\bar{\text{R}}$T *and* SBWPS$\bar{\text{R}}$T, *we call them* signed prefix and suffix reversal and transposition breakpoints, *or* psrt-breakpoints, *and* $1 \leq i < n$. *Note that neither* (π_0, π_1) *nor* (π_n, π_{n+1}) *are psrt-breakpoints. We also impose, by definition, that* $(-1, -n)$ *and* $(n, 1)$ *are never psrt-breakpoints.*

We denote the number of g-breakpoints, p-breakpoints, ps-breakpoints, and psrt-breakpoints in a permutation π as $b_g(\pi)$, $b_p(\pi)$, $b_{ps}(\pi)$, and $b_{psrt}(\pi)$, respectively. We can easily see that $b_g(\iota_n) = b_p(\iota_n) = b_{ps}(\iota_n) = b_{psrt}(\iota_n) = 0$, $b_{ps}(\bar{\eta_n}) = b_{psrt}(\bar{\eta_n}) = 0$, and $b_{psrt}(\varphi^a_{k,n}) = b_{psrt}(\varphi^s_{k,n}) = 0$ for any $2 \leq k \leq n$. Note that these are the only permutations with zero breakpoints in the signed/transposition types. Also note that $b_{ps}(\eta_n) = n - 1$.

Example 6. *Next we show, for each of the four types of breakpoints defined above, how to count them over the same four permutations (two of them are the same as the previous example). A breakpoint is indicated by a "*.*":*

g-breakpoints:	(1 . 4 . 3 . 8 . 5 6 . 2 . 7 .)	(. 5 6 . 2 . 8 . 1 . 4 . 3 . 7 .)
p-breakpoints:	(1 . 4 . 3 . 8 . 5 6 . 2 . 7 .)	(5 6 . 2 . 8 . 1 . 4 . 3 . 7 .)
ps-breakpoints:	(1 . 4 . 3 . 8 . 5 6 . 2 . 7)	(5 6 . 2 . 8 . 1 . 4 . 3 . 7)
psrt-breakpoints:	(1 . 4 . 3 . 8 . 5 6 . 2 . 7)	(5 6 . 2 . 8 1 . 4 . 3 . 7)

g-breakpoints:	(1 . −4 −3 . 8 . −5 . 6 . 2 . −7 .)	(. 5 6 . −2 . −8 . −1 . 4 . 3 . 7 .)
p-breakpoints:	(1 . −4 −3 . 8 . −5 . 6 . 2 . −7 .)	(5 6 . −2 . −8 . −1 . 4 . 3 . 7 .)
ps-breakpoints:	(1 . −4 −3 . 8 . −5 . 6 . 2 . −7)	(5 6 . −2 . −8 . −1 . 4 . 3 . 7)
psrt-breakpoints:	(1 . −4 −3 . 8 . −5 . 6 . 2 . −7)	(5 6 . −2 . −8 . −1 . 4 . 3 . 7)

Note that, when prefix rearrangements are allowed in the rearrangement model, (π_0, π_1) is never a breakpoint and, when suffix rearrangements are allowed, (π_n, π_{n+1}) is never a breakpoint. This happens because prefix (resp. suffix) rearrangements are always moving element π_1 (resp. π_n) and because π_1 can be 1 (resp. π_n can be n) several times before the end of the sorting, so it is not worth considering these positions when we count the number of breakpoints.

We define an *x-move* as a single rearrangement that removes x breakpoints from a permutation. Because of the number of pairs of elements that each rearrangement affects, it is easy to see that a prefix or a suffix reversal can be a 1-move, a prefix or a suffix transposition or a reversal can be a 1 or a 2-move, and a transposition can be a 1, a 2, or a 3-move. All of them can be 0-moves.

Definition 13. *A* strip *is a sequence* $\langle \pi_i \ \ldots \ \pi_j \rangle$ *of elements of* π, *with* $1 \leq i \leq j \leq n$, *such that (i) either* $i = 1$ *or* (π_{i-1}, π_i) *is a breakpoint; (ii) either* $j = n$ *or* (π_j, π_{j+1}) *is a breakpoint; and (iii) no consecutive elements of the sequence are breakpoints.*

For unsigned permutations, a strip of length greater than or equal to two is *increasing* if $\pi_{k+1} = \pi_k + 1$ for all $i \leq k < j$ (in the case of SBPSRT or SBWPSRT, also when $\pi_{k+1} = n$ while $\pi_k = 1$), and it is *decreasing* if $\pi_{k+1} = \pi_k - 1$ (in the case of SBPSRT or SBWPSRT, also when $\pi_{k+1} = 1$ and $\pi_k = n$). Note that, when considering the problems SBPT, SBPST, SBWPT, and SBWPST, the permutations can only have increasing strips. For signed permutations, we only differentiate *positive* or *negative* strips, when they contain only positive or negative elements, respectively. In any case, a strip with one element is called a *singleton*. Whenever it is convenient, we will underline the strips of a permutation.

Example 7. *Next we show, for each of the eight types of breakpoints defined, how to identify strips over some permutations. The strips are underlined:*

gr-breakpoints:	$(\underline{1}\ \underline{4\ 3}\ \underline{8}\ \underline{5\ 6}\ \underline{2}\ \underline{7})$	$(\underline{5\ 6}\ \underline{2}\ \underline{8}\ \underline{1}\ \underline{4\ 3}\ \underline{7})$	
upr-breakpoints:	$(\underline{1}\ \underline{4\ 3}\ \underline{8}\ \underline{5\ 6}\ \underline{2}\ \underline{7})$	$(\underline{5\ 6}\ \underline{2}\ \underline{8}\ \underline{1}\ \underline{4\ 3}\ \underline{7})$	
upsr-breakpoints:	$(\underline{1}\ \underline{4\ 3}\ \underline{8}\ \underline{5\ 6}\ \underline{2}\ \underline{7})$	$(\underline{5\ 6}\ \underline{2}\ \underline{8}\ \underline{1}\ \underline{4\ 3}\ \underline{7})$	
upsrt-breakpoints:	$(\underline{1}\ \underline{4\ 3}\ \underline{8}\ \underline{5\ 6}\ \underline{2}\ \underline{7})$	$(\underline{5\ 6}\ \underline{2}\ \underline{8\ 1}\ \underline{4\ 3}\ \underline{7})$	
g-breakpoints:	$(\underline{1}\ \underline{4}\ \underline{3}\ \underline{8}\ \underline{5\ 6}\ \underline{2}\ \underline{7})$	$(\underline{5\ 6}\ \underline{2}\ \underline{8}\ \underline{1}\ \underline{4}\ \underline{3}\ \underline{7})$	$(\underline{1}\ \underline{-4\ -3}\ \underline{8}\ \underline{-5}\ \underline{6}\ \underline{2}\ \underline{-7})$
p-breakpoints:	$(\underline{1}\ \underline{4}\ \underline{3}\ \underline{8}\ \underline{5\ 6}\ \underline{2}\ \underline{7})$	$(\underline{5\ 6}\ \underline{2}\ \underline{8}\ \underline{1}\ \underline{4}\ \underline{3}\ \underline{7})$	$(\underline{1}\ \underline{-4\ -3}\ \underline{8}\ \underline{-5}\ \underline{6}\ \underline{2}\ \underline{-7})$
ps-breakpoints:	$(\underline{1}\ \underline{4}\ \underline{3}\ \underline{8}\ \underline{5\ 6}\ \underline{2}\ \underline{7})$	$(\underline{5\ 6}\ \underline{2}\ \underline{8}\ \underline{1}\ \underline{4}\ \underline{3}\ \underline{7})$	$(\underline{1}\ \underline{-4\ -3}\ \underline{8}\ \underline{-5}\ \underline{6}\ \underline{2}\ \underline{-7})$
psrt-breakpoints:	$(\underline{1}\ \underline{4}\ \underline{3}\ \underline{8}\ \underline{5\ 6}\ \underline{2}\ \underline{7})$	$(\underline{5\ 6}\ \underline{2}\ \underline{8\ 1}\ \underline{4}\ \underline{3}\ \underline{7})$	$(\underline{1}\ \underline{-4\ -3}\ \underline{8}\ \underline{-5}\ \underline{6}\ \underline{2}\ \underline{-7})$

2.2 Binary Strings

Bender *et al.* [5] showed that binary strings, which are strings that contain only the symbols 0 and 1, are closely related to sorting permutations by length-weighted rearrangements. For this reason, we also consider results for sorting binary strings by length-weighted rearrangements in Chapters 5, 6, and 7. In this section we present some important concepts that will be necessary only in those chapters.

Definition 14. *An* unsigned binary string *of size n is represented as $T = t_1 t_2 \dots t_n$, where $t_i \in \{0, 1\}$ for all $1 \leq i \leq n$.*

Definition 15. *A* signed binary string *of size n is represented as $T = t_1 t_2 \dots t_n$, where $t_i \in \{-1, -0, 0, 1\}$ for all $1 \leq i \leq n$.*

We also always consider the *extended* binary string, which has $t_0 = 0$ and $t_{n+1} = 1$ as fixed elements.

A maximal contiguous substring of only 0's or only 1's is called a *block* of a string T and the *weight* of a block is the number of bits in it. We will assume that the leading block in a binary string is a block of 0's and that the closing block is a block of 1's. Therefore, a binary string T with $g + 1$ blocks of 0's and $g + 1$ blocks of 1's can be represented as a string of blocks $T = 0^{w_0} 1^{w_1} \dots 0^{w_{2g}} 1^{w_{2g+1}}$, meaning that T has w_0 0's, followed by w_1 1's, followed by w_2 0's, and so on.

Definition 16. *A binary string T is* sorted *if it consists of a block of 0's followed by a block of 1's.*

Reversals and transpositions over binary strings are defined exactly as they are defined for permutations. A reversal $\rho(i,j)$ acting on a string T transforms it into $T \cdot \rho(i,j) = t_1 \dots t_{i-1}\underline{t_j \dots t_i}t_{j+1} \dots t_n$, a signed reversal $\bar{\rho}(i,j)$ acting on a signed binary string T transforms it into $T \cdot \bar{\rho}(i,j) = t_1 \dots t_{i-1}\underline{-t_j \dots - t_i}t_{j+1} \dots t_n$, and a transposition $\tau(i,j,k)$ transforms T (signed or unsigned) into $T \cdot \tau(i,j,k) = t_1 \dots t_{i-1}\underline{t_j \dots t_{k-1}}\ \underline{t_i \dots t_{j-1}}t_k \dots t_n$. Prefix reversals, suffix reversals, prefix transpositions, suffix transpositions, and rearrangement model also have the same definition.

Definition 17. *Consider a rearrangement model β and some $\alpha \geq 0$. The* distance *of a binary string T is defined as the cost of a sorting sequence for T of minimum cost and it is denoted as $c^{\alpha}_{\beta}(T)$.*

Definition 18. *Consider a rearrangement model β and some $\alpha \geq 0$. The* diameter *of a problem of sorting binary strings is the biggest distance among the distances of all binary strings of size n and it is denoted as $C^{\alpha}_{\beta}(n)$. Specifically, $C^{\alpha}_{\beta}(n) = \max\{c^{\alpha}_{\beta}(T) : T \text{ has size } n\}$.*

We consider that the median of a set $\{a, a+1, a+2, \dots, a+k\}$ of integers is $a - 1 + \lceil (a+k-a+1)/2 \rceil$.

Definition 19. *A permutation π of size n is* partitioned *if all elements smaller than or equal to (resp. greater than) the median $\lceil n/2 \rceil$ of $[1..n]$ are located to its left (resp. right).*

Note that an idea to sort a permutation π is to first partition it and then recursively sort each of the halves. To perform the partition, we can map $\pi = (\pi_1\ \pi_2\ \dots\ \pi_n)$ to a binary string $T = t_1 t_2 \dots t_n$ such that, for all $1 \leq i \leq n$,

$$t_i = \begin{cases} 0 & \text{if } \pi_i \leq \lceil \frac{n}{2} \rceil \\ 1 & \text{otherwise} \end{cases}$$

and then sort T.

More generally, we can define a function $M(n,p,\pi)$, where n is a positive integer, $p \in \{0,1,2,\dots,n\}$ is called a *pivot*, and π is a permutation of size n, which returns a binary string $T = t_1 t_2 \dots t_n$ of size n such that, for all $1 \leq i \leq n$,

$$t_i = \begin{cases} 0 & \text{if } \pi_i \leq p \\ 1 & \text{otherwise} \end{cases}$$

Let π be a permutation of size n, $T = M(n,p,\pi)$ for some pivot p, β be a rearrangement model, and $\alpha > 0$ be a real number. We have that

$$c^{\alpha}_{\beta}(T) \leq d^{\alpha}_{\beta}(\pi), \tag{2.1}$$

because a sorting sequence for π also sorts T, but T could probably be sorted with a smaller cost.

Example 8. *Let π = (3 8 4 1 2 5 9 6 10 7), $T = M(10,6,\pi) = 0100001011$, and $T' = M(10,4,\pi) = 0100011111$. Note that $\rho(9,10)$, $\rho(7,9)$, $\rho(2,6)$, $\rho(6,8)$, $\rho(2,5)$, $\rho(1,4)$,*

$\rho(1,2)$, $\rho(3,4)$ is a valid sorting sequence of reversals for π. It also clearly sorts T'. However, a unique reversal $\rho(2,5)$ sorts T'. Furthermore, it is easy to see that any sorting sequence of reversals for π has to have more than one reversal.

Note that, through the transformation M, all permutations of size n can build all binary strings of size n if we take all possible values of p. For instance, the identity permutation $\iota_n = (1\ 2\ \ldots\ n)$ itself can generate the $n+1$ binary strings 1^n (when $p = 0$), 01^{n-1} (when $p = 1$), $0^2 1^{n-2}$ (when $p = 2$), and so on, until 0^n (when $p = n$). The binary string $0^2 1^{n-2}$, on the other hand, can also be generated by any permutation that starts with 1 2 or 2 1 when $p = 2$.

Since all permutations of size n can build all binary strings of size n, as stated above, and because of Equation (2.1), we have that

$$C_\beta^\alpha(n) \leq D_\beta^\alpha(n). \tag{2.2}$$

Therefore, any lower bound for $C_\beta^\alpha(n)$ is also a lower bound for $D_\beta^\alpha(n)$ and any upper bound for $D_\beta^\alpha(n)$ is also an upper bound for $C_\beta^\alpha(n)$.

2.3 Approximation Algorithms

Many sorting by genome rearrangements problems are NP-hard. Therefore, in many cases it is good enough to find solutions that are close to some optimal one. Approximation algorithms can find such solutions and guarantee how close they are [19].

Definition 20. *Let ALG be a polynomial-time algorithm on the size of the input I for a minimization problem. Let $ALG(I)$ and $OPT(I)$ be the cost used by the algorithm and the cost of an optimal solution for instance I, respectively. We say ALG is an a-approximation algorithm for the problem if*

$$ALG(I) \leq a \times OPT(I)$$

for all instances I. Note that $OPT(I) \leq ALG(I)$ always.

Chapter 3

Known Results for Traditional Approach

This chapter presents a review of the existing results, to the best of our knowledge, for the traditional approach, where the cost of sorting a permutation is given by the amount of rearrangements that were used to do it. We consider related work only, namely those in which the rearrangement model involves general or restricted versions of reversals and/or transpositions.

One of the most studied problems in Genome Rearrangements is the problem of Sorting by Reversals (SBR), for which there exist several results and approximation algorithms. In 1993, Kececioglu and Sankoff [41] presented the first approximation algorithm, of factor 2, which consists in removing the largest possible amount of breakpoints prioritizing the reversals that leave decreasing strips. Then, Bafna and Pevzner [2] introduced the idea of breakpoint graphs, being able to give two other approximation algorithms of factor 1.8 and 1.75. In 1998, Christie [16] introduced the reversal graph and improved the approximation factor to 1.5. At last, in 2002 Berman *et al.* [7] presented the algorithm that has the best-known approximation factor so far, of 1.375. In the mean time, SBR was proved to be NP-hard [11] and not approximable within a factor of 1.0008.

Bafna and Pevzner [2], with the introduction of breakpoint graphs, were also capable of proving a conjecture that they credit to Holger Golan, which says that $D_r(n) = n - 1$ and that the only permutations with such reversal distance are $\gamma^{(n)}$ and its inverse, where

$$\gamma^{(n+1)} = \begin{cases} (1) & \text{if } n \text{ is zero,} \\ (\gamma_1^{(n)}\ \gamma_2^{(n)}\ \ldots\ \gamma_{n-1}^{(n)}\ n+1\ \gamma_n^{(n)}) & \text{if } n \text{ is odd,} \\ (\gamma_1^{(n)}\ \gamma_2^{(n)}\ \ldots\ \gamma_{n-2}^{(n)}\ n+1\ \gamma_n^{(n)}\ \gamma_{n-1}^{(n)}) & \text{if } n \text{ is even.} \end{cases}$$

On the other hand, the problem of Sorting by Signed Reversals (SB$\bar{\text{R}}$) is polynomially solvable, as shown by Hannenhalli and Pevzner [37] who presented a $O(n^4)$ algorithm. The theory involving such algorithm was explained in a much simpler manner by Bergeron [6]. Also, improvements have been made ever since to decrease this time complexity so that the best-known is $O(n^{3/2})$ [36]. Furthermore, there is a $O(n)$ algorithm which is capable of computing the sorting distance without showing the sorting sequence [1]. The diameter of SB$\bar{\text{R}}$ is, therefore, known to be $n + 1$ [28].

Another important problem in Genome Rearrangements is Sorting by Transpositions (SBT). It was introduced in 1998 by Bafna and Pevzner [3], which gave three approximation algorithms for it, with factors 2, 1.75, and 1.5. The best-known algorithm, though, also has an approximation factor of 1.375 and it was given in 2006 by Elias and Hartman [26]. SBT was proved to be NP-hard in 2011 [9]. It is also NP-hard to decide if a permutation π can be sorted with exactly $b_g(\pi)/3$ transpositions.

Although the diameter of SBT is unknown, there are some results about it. Bafna and Pevzner [3] showed that $\lfloor n/2 \rfloor \leq D_t(n) \leq 3n/4$. The best lower and upper bounds are, respectively, $\lfloor (n+1)/2 \rfloor + 1$ [26], for $n \geq 1$, and $\lfloor (2n-2)/3 \rfloor$, for $n \geq 9$ [27]. In addition, Christie [16], Eriksson *et al.* [27], Meidanis *et al.* [48] showed that $d_t(\eta_n) = \lfloor n/2 \rfloor + 1$.

Walter *et al.* [56] considered problems for which more than one type of rearrangement is allowed, namely Sorting by Reversals and Transpositions (SBRT) and Sorting by Signed Reversals and Transpositions (SB$\bar{\text{R}}$T). They gave a 3-approximation algorithm for the former and a 2-approximation algorithm for the latter. The best-known algorithm for SBRT, however, is a $2k$-approximation [51], where k is the approximation factor of the algorithm for maximizing the number of cycles in a cycle decomposition of the cycle graph [3]. The best-known value for k is $1.4167 + \epsilon$ [12] for any positive ϵ.

Regarding unsigned permutations, Gu *et al.* [35] considered a third event along with reversals and transpositions: the *transreversal*, which is a transposition where one of the segments is reversed. They gave a 2-approximation algorithm when the three are allowed. Lin and Xue [43] considered yet a fourth event that they called *revrev*, which consists in a transposition where the two segments are reversed. When considering the four events, they gave a 1.75-approximation algorithm but currently the best-known approximation factor is 1.5 [38].

One can notice that in the first description of the Pancake Flipping problem, or the Sorting by Prefix Reversals problem (SBPR), the concern was to find its diameter. In fact, the first computational results for the Pancake Flipping problem were given by Gates and Papadimitriou [34], who showed that $D_{pr}(n)$ lies between $17n/16$ and $(5n+5)/3$. Heydari and Sudborough [40] improved the lower bound only eighteen years later to $15n/14$, the best-known so far. In 2009, the upper bound was improved to $18n/11 + O(1)$ [15], also the best-known so far. SBPR was proved to be NP-hard by Bulteau *et al.* [10] and the best-known approximation algorithm for it has factor 2, given by Fischer and Ginzinger [30]. It is also NP-hard to decide if a given permutation π can be sorted by $b_{upr}(\pi)$ prefix reversals.

The Sorting by Signed Prefix Reversals problem (SBP$\bar{\text{R}}$), or the Burnt Pancake Flipping problem, was introduced by Gates and Papadimitriou [34]. They were also concerned about the diameter, and they showed that it lies between $3n/2 - 1$ and $2n + 3$. However, the best-known lower and upper bounds are, respectively, $(3n+3)/2$ [40] for any n, and $2n - 6$ [17] for $n \geq 16$. Cohen and Blum [18] gave the best-known approximation algorithm for this problem and it has factor 2. The complexity of SBP$\bar{\text{R}}$ remains open.

In 2002, Dias and Meidanis [24] introduced the Sorting by Prefix Transpositions problem (SBPT), and gave a 2-approximation algorithm for it, which is the best-known factor to the moment. The diameter of this problem is unknown, but it lies between $\lfloor (3n+1)/4 \rfloor$ [42], for $n \geq 2$, and $n - \log_{7/2} n$ [13]. Also, the complexity of the problem

remains open.

Hasan *et al.* [39] considered Sorting by Prefix Reversals and Transpositions (SbPRT), in which prefix reversals and prefix transpositions are allowed simultaneously, and gave a 3-approximation algorithm with no considerations regarding the diameter. They also considered Sorting by Prefix Reversals and Prefix Transreversals, presenting a 2-approximation algorithm for it. Currently, the best-known algorithm for SbPRT has an approximation factor of $2 + 4/b_{upr}(\pi)$ [23].

3.1 Summary of the Chapter

Table 3.1 summarizes the best-known approximation factors and the best-known bounds for the diameters of the problems that were mentioned throughout this chapter.

Table 3.1: Summary of best-known results for genome rearrangement problems. A '-' indicates that there is no known-result.

Rearrangements	**Best Approx. Factor**	**Complexity**	**Diameter**	
			Lower Bound	**Upper Bound**
Reversals	1.375 [7]	NP-hard [11]	$n-1$ [2]	
Sig. Reversals	1 [37]	P [37]	$n+1$ [28]	
Transpositions	1.375 [26]	NP-hard [9]	$\lfloor \frac{n+1}{2} \rfloor + 1$ [26]	$\lfloor \frac{2n-2}{3} \rfloor$ [27]
Reversals and Transpositions	2.8334 [12, 51]	Unknown	-	-
Sig. Reversals and Transpositions	2 [56]	Unknown	-	-
Pref. Reversals	2 [30]	NP-hard [10]	$\frac{15n}{14}$ [40]	$\frac{18n}{11} + O(1)$ [15]
Sig. Pref. Reversals	2 [18]	Unknown	$\frac{3n+3}{2}$ [40]	$2n-6$ [17]
Pref. Transpositions	2 [24]	Unknown	$\lfloor \frac{3n+1}{4} \rfloor$ [42]	$n - \log_{7/2} n$ [13]
Pref. Reversals and Transpositions	$2 + 4/b_{upr}(\pi)$ [23]	Unknown	-	-

Chapter 4

Results Obtained for Traditional Approach

This chapter, as the previous one, also considers the traditional approach, where the sorting distance of a permutation is given by the amount of rearrangements that were used to sort it, but it presents the results that we obtained.

We start by noticing that the identity permutation has the smallest number of breakpoints (although, as mentioned before, it is not always the only one that has the smallest number). Therefore, we can see the sorting of a permutation π as a reduction of its number of breakpoints. This notion allows us to establish lower bounds on the rearrangement distances.

Lemma 1. *For any unsigned permutation π,*

$$d_{psr}(\pi) \geq b_{upsr}(\pi),\ d_{pst}(\pi) \geq \left\lceil \frac{b_{ps}(\pi)}{2} \right\rceil,\ \textit{and}\ d_{psrt}(\pi) \geq \left\lceil \frac{b_{upsrt}(\pi)}{2} \right\rceil.$$

Proof. A prefix reversal $\rho_p(i)$ separates the pairs of elements (π_0, π_1) and (π_i, π_{i+1}), which can reduce the number of breakpoints by at most one unit, since (π_0, π_1) is never a upsr-breakpoint or a upsrt-breakpoint. A prefix transposition, on the other hand, separates the pairs (π_0, π_1), (π_{i-1}, π_i) and (π_{j-1}, π_j), which can reduce the number of breakpoints by at most two units, since (π_0, π_1) is never a ps-breakpoint or a upsrt-breakpoint. A similar argument applies for a suffix reversal $\rho_s(j)$ and a suffix transposition $\tau_s(j, k)$. Therefore, if we always manage to remove the highest possible amount of breakpoints, we would obtain the stated lower bounds as the distance. □

Using arguments similar to those developed in the proof of Lemma 1, we can also derive lower bounds for the distances in which signed permutations are considered.

Lemma 2. *For any signed permutation π,*

$$d_{ps\bar{r}}(\pi) \geq b_{ps}(\pi),\ d_{p\bar{r}t}(\pi) \geq \left\lceil \frac{b_{p}(\pi)}{2} \right\rceil,\ \textit{and}\ d_{ps\bar{r}t}(\pi) \geq \left\lceil \frac{b_{psrt}(\pi)}{2} \right\rceil.$$

We will call `2-PR`, `2-PT`, `2-PRT`, and `2-PR̄` the best-known algorithms that exist so far in the literature for the problems SbPR [30], SbPT [24], SbPRT [23], and SbPR̄ [18],

respectively, which were mentioned in Chapter 3. It was proven that these algorithms guarantee that

$$\begin{gathered}\texttt{2-PR}(\pi, n) \leq 2b_{upr}(\pi),\\ \texttt{2-PT}(\pi, n) \leq b_p(\pi) - 1,\\ \texttt{2-PRT}(\pi, n) \leq b_{upr}(\pi) + 2, \text{ and}\\ \texttt{2-P}\bar{\texttt{R}}(\pi, n) \leq 2b_p(\pi).\end{gathered}$$

We first point out that they are also valid algorithms for SbPSR, SbPST, SbPSRT, and SbPS$\bar{\text{R}}$, respectively. Second, by our definition of breakpoints, when suffix rearrangements are allowed (which means (π_n, π_{n+1}) is not a breakpoint), we have

$$\begin{gathered}b_{upr}(\pi) - 1 \leq b_{upsr}(\pi) \leq b_{upr}(\pi),\\ b_{upr}(\pi) - 1 \leq b_{upsrt}(\pi) \leq b_{upr}(\pi), \text{ and}\\ b_p(\pi) - 1 \leq b_{ps}(\pi) \leq b_p(\pi).\end{gathered}$$

This means that these four algorithms guarantee that

$$\begin{gathered}\texttt{2-PR}(\pi, n) \leq 2b_{upsr}(\pi) + 2,\\ \texttt{2-PT}(\pi, n) \leq b_{ps}(\pi),\\ \texttt{2-PRT}(\pi, n) \leq b_{upsrt}(\pi) + 3, \text{ and}\\ \texttt{2-P}\bar{\texttt{R}}(\pi, n) \leq 2b_{ps}(\pi) + 2.\end{gathered}$$

Therefore, considering the lower bound given by Lemma 1, for instance, algorithm 2-PR has an approximation factor of $(2b_{upsr}(\pi)+2)/b_{upsr}(\pi)$ for SbPSR, which is $2+2/b_{upsr}(\pi)$. The same analysis can be made for the other three algorithms: using the upper bounds given above along with the lower bounds given by Lemmas 1 and 2, 2-PRT and 2-P$\bar{\text{R}}$ are $(2 + 6/b_{upsrt}(\pi))$-approximation and $(2 + 2/b_{ps}(\pi))$-approximation for SbPSRT and SbPS$\bar{\text{R}}$, respectively, and 2-PT is a 2-approximation for SbPST.

In Sections 4.1 to 4.6 we will present specific algorithms for SbPSR, SbPST, SbPSRT, and SbPS$\bar{\text{R}}$, and also for SbP$\bar{\text{R}}$T and SbPS$\bar{\text{R}}$T, thanks to which we are able to decrease the upper bounds given above. Furthermore, they take advantage of suffix rearrangements, which makes them return better results, in practice, than just using the prefix ones. In Section 4.7 we present some experimental results on these algorithms. In Section 4.8, we additionally present bounds for the diameters of these problems. Lastly, in Section 4.9, we give a summary of all results presented in this chapter.

4.1 Sorting by Prefix and Suffix Reversals

In this section we present an algorithm for SbPSR problem, which was recently proved to be NP-hard by Fertin *et al.* [29]. The general idea of our algorithm is, while the permutation is not sorted: try to apply a 1-move or else a 1-move followed by a 0-move; if neither is possible, the permutation has a specific format that can be sorted with at most its number of breakpoint operations.

We define now the *breakpoint graph* [2] of an unsigned permutation π, which is used by 2-PR and it will be used in this section for SbPSR. The breakpoint graph is the graph

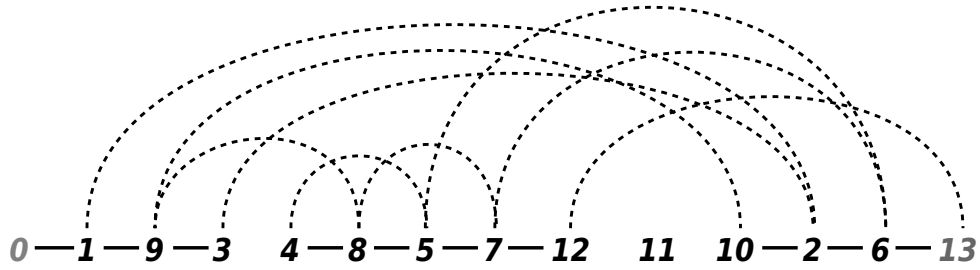
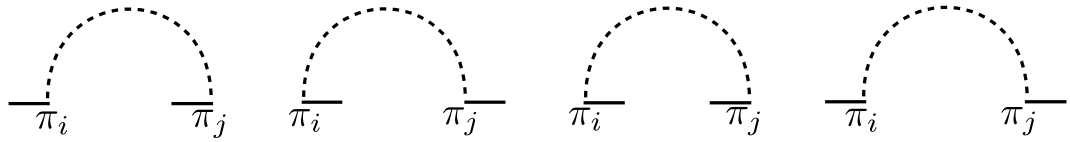


Figure 4.1: Breakpoint graph for $\pi = (1\ 9\ 3\ 4\ 8\ 5\ 7\ 12\ 11\ 10\ 2\ 6)$ and considering SBPSR. Note that edge $(0, 1)$ must always exist if prefix rearrangements are involved and edge $(6, 13)$ must always exist if suffix rearrangements are involved.



(a) Type 1. (b) Type 2. (c) Type 3. (d) Type 4.

Figure 4.2: Classification of gray edges.

$G(\pi) = (V, E)$ in which $V = \{\pi_0, \pi_1, \ldots, \pi_{n+1}\}$ and E contains *black edges* and *gray edges*. A black edge e exists if and only if (i) $e = (\pi_i, \pi_{i+1})$ and π_i and π_{i+1} form a breakpoint, for $0 \leq i \leq n$; (ii) $e = (\pi_0, \pi_1)$ and the model includes prefix rearrangements; and (iii) $e = (\pi_n, \pi_{n+1})$ and the model includes suffix rearrangements. A gray edge e exists if and only if $e = (\pi_i, \pi_j)$ for some i and j such that $0 \leq i < j \leq n + 1$ with $\pi_j = \pi_i \pm 1$ and $j \neq i + 1$. In a graphic representation, we draw black edges as straight lines and gray edges as dashed lines, as shown by Figure 4.1.

Let (π_i, π_j) be any edge of $G(\pi)$. Although the graph is not directed, we will always consider that $i < j$ so that we can say that such edge *begins* at π_i (or at position i) and *ends* at π_j (or at position j). If (π_i, π_j) is a gray edge, then we have $\pi_j = \pi_i \pm 1$, which means that there must be at least one black edge that begins or ends at π_i as well as there must be at least one black edge that begins or ends at π_j. Hence, we can classify (π_i, π_j) into at least one of the four types showed in Figure 4.2.

Lemma 3 defines what we call *good edges*: when they exist, it is always possible to apply a 1-move or else a 0-move followed by a 1-move.

Lemma 3. *Let π be any unsigned permutation. There is a sequence of at most two prefix reversals or suffix reversals that removes one upsr-breakpoint if $G(\pi)$ contains at least one gray edge (π_i, π_j):*

1. *of type 1 with $i = 1$ and $j \leq n$;*
2. *of type 2 with $j = n$ and $i \geq 1$;*
3. *of type 3 with $i = 1$ and $j \leq n$;*
4. *of type 3 with $j = n$ and $i \geq 1$;*
5. *of type 2 with $i \geq 1$ and $j \leq n$;*

6. *of type 1 with* $j \leq n$ *and* $i \geq 1$*;*

7. *of type 3 with* $i > 1$ *and* $j \leq n$*;*

8. *of type 3 with* $j < n$ *and* $i \geq 1$*.*

These eight categories of gray edges are called good edges*. We call* good prefix edges *(GPE) the ones presented on items 1, 3, 5, and 7. The others are called* good suffix edges *(GSE).*

Proof. If (π_i, π_j) is a gray edge, then $\pi_j = \pi_i \pm 1$. In order to create an adjacency between π_i and π_j without creating new upsr-breakpoints, one can perform, for each category:

1. one prefix reversal $\rho_p(j-1)$;
2. one suffix reversal $\rho_s(i+1)$;
3. one prefix reversal $\rho_p(j-1)$;
4. one suffix reversal $\rho_s(i+1)$;
5. two prefix reversals $\rho_p(j)$ and $\rho_p(j-i)$;
6. two suffix reversals $\rho_s(i)$ and $\rho_s(n+1-(j-i))$;
7. two prefix reversals $\rho_p(i)$ and $\rho_p(j-1)$;
8. two suffix reversals $\rho_s(j)$ and $\rho_s(i+1)$.

□

We point out that algorithm `2-PR` only deals with good prefix edges without considering the constraints over j.

We then propose algorithm `2-PSR`, which searches for any of the eight categories of good edges given by Lemma 3, in that order. In our implementation, the algorithm scans the permutation from right to left when searching for good prefix edges and from left to right when searching for good suffix edges; when considering a certain category, it will use the first edge it finds. When a permutation does not contain a good edge, it is characterized by Lemmas 4 and 5, and we can transform it into ι_n with at most $b_{upsr}(\pi)+2$ reversals, as Lemma 6 shows.

Lemma 4. *For* SbPSR*, if* $\pi \neq \iota_n$ *is an unsigned permutation that does not contain a good edge, then* π *does not contain any singleton.*

Proof. Suppose that π contains a singleton $\pi_i = k$. Note that a black edge ends at k and another black edge starts at k. In addition, two gray edges also start or end at k. If $k+1$ and $k-1$ were to the right of k, then there would exist good prefix edges of types 2 or 3 in π, a contradiction. If $k+1$ or $k-1$ were to the left of k, then there would be good suffix edges of types 1 or 3 in π, also a contradiction. □

Lemma 5. *For* SbPSR*, if $\pi \neq \iota_n$ is an unsigned permutation that does not contain a good edge, then π is of one of the three forms:*

1. $\eta_n = (n\ n{-}1\ n{-}2\ \ldots\ 1)$;
2. $\delta^a_{b+1} = (\underbrace{p_1\ p_1{-}1\ \ldots\ 1}_{\ell_1}\ \underbrace{p_2\ p_2{-}1\ \ldots\ p_1{+}1}_{\ell_2}\ \ldots\ldots\ \underbrace{n\ n{-}1\ \ldots\ p_b{+}1}_{\ell_{b+1}})$;
3. $\delta^d_{b+1} = (\underbrace{p_b{+}1\ p_b{+}2\ \ldots\ n}_{\ell_1}\ \ldots\ldots\ \underbrace{p_1{+}1\ p_1{+}2\ \ldots\ p_2}_{\ell_b}\ \underbrace{1\ 2\ \ldots\ p_1}_{\ell_{b+1}})$,

where $b = b_{upsr}(\pi)$ and $\ell_i \geq 2$ for all $1 \leq i \leq b+1$.

Proof. If $\pi = \eta_n$, then π has only two black edges $(0, n)$ and $(1, n+1)$ and two gray edges $(0, 1)$ and $(n, n+1)$, which are not considered good edges, due to the restrictions over the indices, as given in Lemma 3.

We will show by induction on the number s of strips of π that if π does not contain a good edge and it is not η_n, then it is either δ^a_{b+1} or δ^d_{b+1}. Note that $s = b+1$.

If $s = 2$ then $b_{upsr}(\pi) = 1$, since neither (π_0, π_1) nor (π_n, π_{n+1}) are upsr-breakpoints, by definition. Among all 6 possible forms of permutations with two strips[1], the only ones that do not contain good edges are $(\underline{k\ k{-}1\ \ldots\ 1}\ \underline{n\ n{-}1\ \ldots\ k{+}1})$ and $(\underline{k{+}1\ k{+}2\ \ldots\ n}\ \underline{1\ 2\ \ldots\ k})$, which correspond to δ^a_2 and δ^d_2, respectively.

Suppose that $s \geq 3$ and that every permutation $\pi \neq \eta_n$ without good edges and with $s-1$ strips is either of the form δ^a_{b+1} or of the form δ^d_{b+1}.

Let π be a permutation with s strips without good edges and let π_w be the last element of the first strip of π. Either $\pi_{w-1} > \pi_w$ and $\pi_w = 1$ (otherwise there would be some $\pi_i = \pi_w - 1$ with $i > w$ and a good prefix edge of type 2 or 3 would exist) or $\pi_{w-1} < \pi_w$ and $\pi_w = n$ (otherwise there would be some $\pi_i = \pi_w + 1$ with $i > w$ and a good prefix edge of type 2 or 3 would exist). Note that 0 and $n+1$ do not form good edges with 1 and n, respectively, because prefix and suffix rearrangements are being allowed by the rearrangement model.

Suppose first that $\pi_w = 1$. Let π' be the permutation built from π such that $\pi'_j = \pi_{j+w} - \pi_1$ for all $1 \leq j \leq n-w$. Note that π' has $s-1$ strips. In addition, it is easy to see that if π' contained good edges, then π would have them too. Therefore, by the induction hypothesis, π' is either of the form δ^a_b or δ^d_b. Since π' corresponds to π relabeled without the first strip, which contains the element 1, if $\pi' = \delta^d_b$ then $\pi = (\underline{p_1\ \ldots\ 1}\ \underline{p_b{+}1\ \ldots\ n}\ \ldots\ldots\ \underline{p_1{+}1\ \ldots\ p_2})$. However, in this case a good prefix edge of type 1 would exist, which is not possible. Therefore, $\pi' = \delta^a_b$ and $\pi = \delta^a_{b+1}$.

Suppose now that $\pi_w = n$. Let π' be the permutation built from π such that $\pi'_j = \pi_{j+w}$ for all $1 \leq j \leq n-w$. Note that π' has $s-1$ strips. In addition, it is easy to see that if π' contained good edges, then π would have them too. Therefore, π' is either of the form δ^a_b or δ^d_b. Since π' corresponds to π without the first strip, which contains the element n, if $\pi' = \delta^a_b$ then $\pi = (\underline{p_b{+}1\ \ldots\ n}\ \underline{p_1\ \ldots\ 1}\ \ldots\ldots\ \underline{p_b\ \ldots\ p_{b-1}{+}1})$. However, in this case a good prefix edge of type 1 would exist, which is not possible. Therefore, $\pi' = \delta^d_b$ and $\pi = \delta^d_{b+1}$. □

[1]The 6 permutations with two strips for upsr-breakpoints are $(1\ \ldots\ k\ n\ \ldots\ k{+}1)$, $(k\ \ldots\ 1\ k{+}1\ \ldots\ n)$, $(k\ \ldots\ 1\ n\ \ldots\ k{+}1)$, $(n\ \ldots\ k{+}1\ 1\ \ldots\ k)$, $(k{+}1\ \ldots\ n\ 1\ \ldots\ k)$, and $(k{+}1\ \ldots\ n\ k\ \ldots\ 1)$.

Lemma 6. *Let π be any unsigned permutation without good edges, that is, of one of the forms given by Lemma 5. If $\pi = \eta_n$, then one prefix reversal $\rho_p(n)$ sorts it. Otherwise, at most $b_{upsr}(\pi) + 2$ prefix and suffix reversals sort it.*

Proof. Let $b = b_{upsr}(\pi)$. First consider that $\pi = \delta^a_{b+1}$ and b is an odd number. The $b+1$ reversals

$$\rho_s(\ell_1 + 1) \cdot \rho_p(n - \ell_2) \cdot \rho_s(\ell_3 + 1) \cdot \rho_p(n - \ell_4) \cdot \cdots \cdot \rho_s(\ell_b + 1) \cdot \rho_p(n - \ell_{b+1})$$

transform π into ι_n, as we show next.

Let π^k, $1 \leq k \leq (b-1)/2$, be the permutation we obtain after applying the first $2k$ reversals of the sequence given above, namely $\rho_s(\ell_1+1)\cdot\rho_p(n-\ell_2)\cdot\cdots\cdot\rho_s(\ell_{2k-1}+1)\cdot\rho_p(n-\ell_{2k})$. It can easily be shown by induction that $\pi^k = (\underbrace{p_{2k+1}\ldots p_{2k}+1}_{\ell_{2k+1}}\ \underbrace{p_{2k+2}\ldots p_{2k+1}+1)}_{\ell_{2k+2}}$ $\ldots\ldots$ $\underbrace{n\ldots p_b+1}_{\ell_{b+1}}\ \underbrace{1\ 2\ \ldots\ p_{2k-2}+1\ldots p_{2k-1}\ p_{2k-1}+1\ldots p_{2k}}_{\ell_1+\ell_2+\ldots+\ell_{2k}})$.

Thus, $\pi^{(b-1)/2} = (\underline{p_b\ \ldots\ p_{b-1}+1}\ \underline{n\ \ldots\ p_b+1}\ \underline{1\ 2\ \ldots\ p_{b-1}})$ and $\rho_s(\ell_b+1)\cdot\rho_p(n-\ell_{b+1})$ finally sorts it.

If $\pi = \delta^d_{b+1}$ and b is odd, then one must apply $\rho_p(n)$ to transform it into δ^a_{b+1} followed by the $b+1$ reversals given above.

If $\pi = \delta^d_{b+1}$ and b is an even number, then the $b+1$ reversals

$$\rho_p(n - \ell_{b+1}) \cdot \rho_s(\ell_b + 1) \cdot \rho_p(n - \ell_{b-1}) \cdot \rho_s(\ell_{b-2} + 1) \cdot \cdots \cdot \rho_p(n - \ell_3) \cdot \rho_s(\ell_2 + 1) \cdot \rho_p(n - \ell_1)$$

sort π. This also can be shown by induction.

If $\pi = \delta^a_{b+1}$ and b is even, then one must apply $\rho_p(n)$ to transform it into δ^d_{b+1} followed by the reversals given above. □

One question that arises when we allow more than one rearrangement (in this case, prefix reversals and suffix reversals) is which one should be applied at each iteration of the algorithm, that executes while there is a breakpoint. We decided just to interpolate the choices, as shown in Algorithm 1, which describes `2-PSR`.

Example 9. *The following example shows the execution of* `2-PSR` *over* $\pi = (9\ 6\ 2\ 14\ 10\ 15\ 7\ 12\ 4\ 11\ 3\ 8\ 13\ 1\ 5)$. *Recall that GPE and GSE stand for good prefix edges and good*

suffix edges, respectively.

$\pi = (\underline{9\ 6\ 2\ 14\ 10\ 15\ 7\ 12\ 4\ 11\ 3}\ 8\ 13\ 1\ 5)$ // *(GPE type 1) 1-move to place π_1 next to $\pi_1 - 1$*
$\pi \leftarrow \pi{\cdot}\rho_p(11) = (\underline{3\ 11\ 4\ 12\ 7\ 15\ 10\ 14}\ 2\ 6\ 9\ 8\ 13\ 1\ 5)$ // *(GPE type 1) 1-move to place π_1 next to $\pi_1 - 1$*
$\pi \leftarrow \pi{\cdot}\rho_p(8) = (\underline{14\ 10\ 15\ 7\ 12\ 4\ 11\ 3\ 2\ 6\ 9\ 8}\ 13\ 1\ 5)$ // *(GPE type 1) 1-move to place π_1 next to $\pi_1 - 1$*
$\pi \leftarrow \pi{\cdot}\rho_p(12) = (\underline{8\ 9\ 6\ 2\ 3\ 11\ 4\ 12}\ 7\ 15\ 10\ 14\ 13\ 1\ 5)$ // *(GPE type 1) 1-move to place π_1 next to $\pi_1 - 1$*
$\pi \leftarrow \pi{\cdot}\rho_p(8) = (\underline{12\ 4}\ 11\ 3\ 2\ 6\ 9\ 8\ 7\ 15\ 10\ 14\ 13\ 1\ 5)$ // *(GPE type 1) 1-move to place π_1 next to $\pi_1 - 1$*
$\pi \leftarrow \pi{\cdot}\rho_p(2) = (\underline{4\ 12\ 11}\ 3\ 2\ 6\ 9\ 8\ 7\ 15\ 10\ 14\ 13\ 1\ 5)$ // *(GPE type 1) 1-move to place π_1 next to $\pi_1 - 1$*
$\pi \leftarrow \pi{\cdot}\rho_p(3) = (\underline{11\ 12\ 4\ 3\ 2\ 6\ 9\ 8\ 7\ 15}\ 10\ 14\ 13\ 1\ 5)$ // *(GPE type 1) 1-move to place π_1 next to $\pi_1 - 1$*
$\pi \leftarrow \pi{\cdot}\rho_p(10) = (\underline{15\ 7\ 8\ 9\ 6\ 2\ 3\ 4\ 12\ 11\ 10}\ 14\ 13\ 1\ 5)$ // *(GPE type 1) 1-move to place π_1 next to $\pi_1 - 1$*
$\pi \leftarrow \pi{\cdot}\rho_p(11) = (\underline{10\ 11\ 12\ 4\ 3\ 2\ 6}\ 9\ 8\ 7\ 15\ 14\ 13\ 1\ 5)$ // *(GPE type 1) 1-move to place π_1 next to $\pi_1 - 1$*
$\pi \leftarrow \pi{\cdot}\rho_p(7) = (\underline{6\ 2\ 3\ 4\ 12\ 11\ 10\ 9\ 8\ 7\ 15\ 14\ 13\ 1}\ 5)$ // *(GPE type 1) 1-move to place π_1 next to $\pi_1 - 1$*
$\pi \leftarrow \pi{\cdot}\rho_p(14) = (\underline{1\ 13\ 14\ 15\ 7\ 8\ 9\ 10\ 11\ 12\ 4\ 3\ 2}\ 6\ 5)$ // *(GPE type 2) first, a 0-move*
$\pi \leftarrow \pi{\cdot}\rho_p(13) = (\underline{2\ 3\ 4\ 12\ 11\ 10\ 9\ 8\ 7\ 15\ 14\ 13}\ 1\ 6\ 5)$ // *then, a 1-move to place π_1 next to $\pi_1 - 1$*
$\pi \leftarrow \pi{\cdot}\rho_p(12) = (13\ 14\ 15\ \underline{7\ 8\ 9\ 10\ 11\ 12\ 4\ 3\ 2\ 1\ 6\ 5})$ // *(GSE type 1) 0-move*
$\pi \leftarrow \pi{\cdot}\rho_s(4) = (13\ 14\ 15\ 5\ 6\ \underline{1\ 2\ 3\ 4\ 12\ 11\ 10\ 9\ 8\ 7})$ // *1-move to place π_n next to $\pi_n - 1$*
$\pi \leftarrow \pi{\cdot}\rho_s(6) = (13\ 14\ 15\ \underline{5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 4\ 3\ 2\ 1})$ // *(GSE type 1) 0-move*
$\pi \leftarrow \pi{\cdot}\rho_s(4) = (13\ 14\ 15\ 1\ 2\ 3\ 4\ \underline{12\ 11\ 10\ 9\ 8\ 7\ 6\ 5})$ // *1-move to place π_n next to $\pi_n - 1$*
$\pi \leftarrow \pi{\cdot}\rho_s(8) = (\underline{13\ 14\ 15\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12})$ // *$\pi = \delta^d_{b+1}$ for $b = 1$ (b odd, so apply $\rho_p(n)$ first)*
$\pi \leftarrow \pi{\cdot}\rho_p(15) = (12\ 11\ 10\ 9\ 8\ 7\ 6\ 5\ 4\ 3\ 2\ 1\ \underline{15\ 14\ 13})$ // *apply $\rho_s(\ell_1 + 1)$*
$\pi \leftarrow \pi{\cdot}\rho_s(13) = (\underline{12\ 11\ 10\ 9\ 8\ 7\ 6\ 5\ 4\ 3\ 2\ 1}\ 13\ 14\ 15)$ // *apply $\rho_p(n - \ell_2)$*
$\pi \leftarrow \pi{\cdot}\rho_p(12) = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$

Lemma 7 and Theorem 1 show that the approximation factor of our algorithm is 2.

Lemma 7. *For any unsigned permutation π, 2-PSR$(\pi, n) \leq 2b_{upsr}(\pi) + 1$.*

Proof. Starting from π, our algorithm greedily removes breakpoints (based on finding a GPE or a GSE) until this is not possible anymore. At worse, it uses two prefix or two suffix reversals to remove one breakpoint. Suppose it does this until some permutation π' is reached. We have $\texttt{2-PSR}(\pi, n) \leq 2(b_{upsr}(\pi) - b_{upsr}(\pi')) + \texttt{2-PSR}(\pi', n)$.

If $b_{upsr}(\pi') = 0$, then either $\pi' = \iota_n$ or $\pi' = \eta_n$, which means $\texttt{2-PSR}(\pi, n) \leq 2b_{upsr}(\pi)+1$.

If $b_{upsr}(\pi') \geq 1$, then according to Lemma 6, $\texttt{2-PSR}(\pi', n) \leq b_{upsr}(\pi') + 2$. Therefore, $\texttt{2-PSR}(\pi, n) \leq 2b_{upsr}(\pi) - b_{upsr}(\pi') + 2 \leq 2b_{upsr}(\pi) + 1$. □

Theorem 1. SbPSR *is 2-approximable.*

Proof. First note that 2-PSR has time complexity $O(n^2)$ where n is the size of the permutation, because finding any good edge takes $O(n)$, applying the corresponding rearrangement takes $O(n)$, and the distance is proportional to the number of breakpoints, which is also $O(n)$.

Regarding the approximation factor of 2-PSR, note that if the permutation π does not have upsr-breakpoints, then it is either the identity or the reverse permutation. In any case, 2-PSR uses the optimum amount of reversals to sort π (zero and one, respectively). Therefore, suppose $b_{upsr}(\pi) \geq 1$.

Suppose that it is possible to apply a 1-move on π, generating π'. Since 2-PSR always tries 1-moves first, we have that $\texttt{2-PSR}(\pi, n) \leq 1 + \texttt{2-PSR}(\pi', n)$ and also $b_{upsr}(\pi') = b_{upsr}(\pi) - 1$. By Lemma 7, we know that $\texttt{2-PSR}(\pi', n) \leq 2b_{upsr}(\pi') + 1$. Therefore,

$$\texttt{2-PSR}(\pi, n) \leq 1 + \texttt{2-PSR}(\pi', n) \leq 1 + 2(b_{upsr}(\pi) - 1) + 1 = 2b_{upsr}(\pi).$$

Using Lemma 1, which states that $d_{psr}(\pi) \geq b_{upsr}(\pi)$, the approximation factor is

$$\frac{2b_{upsr}(\pi)}{b_{upsr}(\pi)} = 2.$$

Algorithm 1 A 2-approximation algorithm for SbPSR.

2-PSR(π, n)

Input: permutation π and its size n
Output: number of rearrangements used to sort π

$d \leftarrow 0$
while $\pi \neq \iota_n$ **do** // Apply rules from Lemma 3
  if $G(\pi)$ has a GPE (π_1, π_j) of type 1 **and** $j \leq n$ **then**
    $\pi \leftarrow \pi \cdot \rho_p(j-1)$
    $d \leftarrow d+1$
  else if $G(\pi)$ has a GSE (π_i, π_n) of type 2 **and** $i \geq 1$ **then**
    $\pi \leftarrow \pi \cdot \rho_s(i+1)$
    $d \leftarrow d+1$
  else if $G(\pi)$ has a GPE (π_1, π_j) of type 3 **and** $j \leq n$ **then**
    $\pi \leftarrow \pi \cdot \rho_p(j-1)$
    $d \leftarrow d+1$
  else if $G(\pi)$ has a GSE (π_i, π_n) of type 3 **and** $i \geq 1$ **then**
    $\pi \leftarrow \pi \cdot \rho_s(i+1)$
    $d \leftarrow d+1$
  else if $G(\pi)$ has a GPE (π_i, π_j) of type 2 **and** $i \neq 0$ **and** $j \leq n$ **then**
    $\pi \leftarrow \pi \cdot \rho_p(j) \cdot \rho_p(j-i)$
    $d \leftarrow d+2$
  else if $G(\pi)$ has a GSE (π_i, π_j) of type 1 **and** $j \neq n+1$ **and** $i \geq 1$ **then**
    $\pi \leftarrow \pi \cdot \rho_s(i) \cdot \rho_s(n+1-(j-i))$
    $d \leftarrow d+2$
  else if $G(\pi)$ has a GPE (π_i, π_j) of type 3 **and** $i > 1$ **and** $j \leq n$ **then**
    $\pi \leftarrow \pi \cdot \rho_p(i) \cdot \rho_p(j-1)$
    $d \leftarrow d+2$
  else if $G(\pi)$ has a GSE (π_i, π_j) of type 3 **and** $j < n$ **and** $i \geq 1$ **then**
    $\pi \leftarrow \pi \cdot \rho_s(j) \cdot \rho_s(i+1)$
    $d \leftarrow d+2$
  else if $\pi = \eta_n$ **then**
    $\pi \leftarrow \pi \cdot \rho_p(n)$
    $d \leftarrow d+1$
  else // Then $\pi = \delta^a_{b+1}$ or $\pi = \delta^d_{b+1}$ (Lemma 5)
    $b \leftarrow b_{upsr}(\pi)$
    if $b \bmod 2 \equiv 1$ **then**
      if $\pi = (p_b{+}1 \ \ldots \ n \ p_{b-1} \ \ldots \ p_b \ \ldots \ 1 \ldots p_1) = \delta^d_{b+1}$ **then**
        $\pi \leftarrow \pi \cdot \rho_p(n)$
        $d \leftarrow d+1$
      Let ℓ_i be the size of the ith strip of π
      // Apply the rearrangements from Lemma 6 for δ^a_{b+1} when $b_{upsr}(\pi)$ is odd
      $\pi \leftarrow \pi \cdot \rho_s(\ell_1+1) \cdot \rho_p(n-\ell_2) \cdot \ldots\ldots \cdot \rho_s(\ell_b+1) \cdot \rho_p(n-\ell_{b+1})$
    else
      if $\pi = (p_1 \ \ldots \ 1 \ p_2 \ \ldots \ p_1{+}1 \ \ldots\ldots \ n \ \ldots \ p_b{+}1) = \delta^a_{b+1}$ **then**
        $\pi \leftarrow \pi \cdot \rho_p(n)$
        $d \leftarrow d+1$
      Let ℓ_{b+2-i} be the size of the ith strip of π
      // Apply the rearrangements from Lemma 6 for δ^d_{b+1} when $b_{upsr}(\pi)$ is even
      $\pi \leftarrow \pi \cdot \rho_p(n{-}\ell_1) \cdot \rho_s(\ell_2{+}1) \cdot \ldots \cdot \rho_p(n{-}\ell_{b-1}) \cdot \rho_s(\ell_b{+}1) \cdot \rho_p(n{-}\ell_{b+1})$
    $d \leftarrow d+b+1$
return d

Now suppose that it is not possible to apply a 1-move on π. This means that the lower bound given in Lemma 1 is not tight and $d_{psr}(\pi) \geq b_{upsr}(\pi) + 1$. Using Lemma 7, the approximation factor is

$$\frac{2b_{upsr}(\pi) + 1}{b_{upsr}(\pi) + 1} \leq \frac{2b_{upsr}(\pi) + 2}{b_{upsr}(\pi) + 1} = 2.$$

□

4.2 Sorting by Prefix and Suffix Transpositions

The algorithm that we propose for SbPST is called `2-PST`, and it works as follows. While the permutation is not sorted, it first tries to apply a 2-move and, if this is not possible, it applies a 1-move over the current permutation. Lemmas 8 and 9 analyze the existence of such moves.

Lemma 8. *Let $\pi \neq \iota_n$ be any unsigned permutation. For* SbPST*, there exists at most two 2-moves that can be applied to π.*

Proof. Suppose that $\tau_p(i, j)$ is a 2-move. Note that we then must have $2 \leq i < j \leq n$. Also, $\pi \cdot \tau_p(i, j) = (\pi_i \ldots \pi_{j-1}\ \pi_1 \ldots \pi_{i-1}\ \pi_j \ldots \pi_n)$, where $\pi_{i-1} = \pi_j - 1 \neq \pi_i - 1$ and $\pi_{j-1} = \pi_1 - 1 \neq \pi_j - 1$. It is easy to see that π_1 uniquely determines j and that j uniquely determines i.

Now suppose that $\tau_s(i, j)$ is a 2-move. Again, $2 \leq i < j \leq n$ and $\pi \cdot \tau_s(i, j) = (\pi_1 \ldots \pi_{i-1}\ \pi_j \ldots \pi_n\ \pi_i \ldots \pi_{j-1})$, where $\pi_i = \pi_n + 1 \neq \pi_{i-1} + 1$ and $\pi_j = \pi_{i-1} + 1 \neq \pi_{j-1} + 1$. It is also easy to see that π_n uniquely determines i and that i uniquely determines j. □

Lemma 9. *Let $\pi \neq \iota_n$ be any unsigned permutation. For* SbPST*, it is always possible to apply a 1-move on π.*

Proof. Let π_i be the last element of the first strip of π (remember that when considering SbPST the permutations can only have increasing strips). There must exist another strip in π that starts with some π_j such that $\pi_j = \pi_i + 1$ and $i < j$, that is, π_j is the element that can increase the size of the first strip. If $\pi_i \neq n$, then $\tau_p(i + 1, j)$ is a 1-move. However, if $\pi_i = n$ then $j = n + 1$, which means that $\tau_p(i + 1, j)$ would not remove any ps-breakpoint. In this case, there must also exist a strip that ends with some π_j such that $\pi_j = \pi_1 - 1$, the previous element of the first strip, and so the transposition $\tau_p(i + 1, j + 1)$ is a 1-move.

Let π_j be the first element of the last strip of π. There must exist another strip in π that starts with some element π_i such that $\pi_i = \pi_j - 1$ and $i < j$, that is, π_i is the element that can increase the last strip. If $\pi_j \neq 1$, then $\tau_s(i + 1, j)$ is a 1-move. However, if $\pi_j = 1$ then $i = 0$, which means that $\tau_s(i + 1, j)$ would not remove any ps-breakpoint from π. In this case, there must also exist a strip in π that ends with some π_i such that $\pi_i = \pi_n + 1$, the next element of the last strip, and so the transposition $\tau_s(i, j)$ is a 1-move. □

Regarding implementation issues, since Lemma 9 shows that more than one 1-move is always possible, to break ties we decided to choose the 1-move that involves the least number of elements. Likewise, if there exist two 2-moves, then we also break ties by

choosing the one that involves the least number of elements. Note that other 1-moves can exist in π, apart from the ones described in Lemma 9. However, we decided to consider only those ones, as Algorithm 2 shows.

Example 10. *The following example shows the execution of* `2-PST` *over* $\pi = (9\ 6\ 2\ 14\ 10\ 15\ 7\ 12\ 4\ 11\ 3\ 8\ 13\ 1\ 5)$*:*

$\pi = (\underline{9\ 6\ 2\ 14\ 10\ 15\ 7\ 12}\ \underline{4\ 11\ 3\ 8}\ 13\ 1\ 5)$ // *2-move with prefix (there is a 2-move with suffix)*
$\pi \leftarrow \pi{\cdot}\tau_p(9{,}13) = (4\ 11\ 3\ 8\ 9\ \underline{6\ 2\ 14}\ \underline{10\ 15\ 7\ 12\ 13\ 1\ 5})$ // *2-move with suffix (there is not a 2-move with prefix)*
$\pi \leftarrow \pi{\cdot}\tau_s(6{,}9) = (4\ 11\ 3\ 8\ 9\ 10\ 15\ 7\ 12\ 13\ \underline{1\ 5\ 6\ 2}\ \underline{14})$ // *1-move with suffix (involves less elements)*
$\pi \leftarrow \pi{\cdot}\tau_s(11{,}15) = (4\ 11\ \underline{3\ 8\ 9\ 10\ 15\ 7}\ \underline{12\ 13\ 14\ 1\ 5\ 6\ 2})$ // *2-move with suffix*
$\pi \leftarrow \pi{\cdot}\tau_s(3{,}9) = (\underline{4}\ \underline{11\ 12\ 13\ 14\ 1}\ 5\ 6\ 2\ 3\ 8\ 9\ 10\ 15\ 7)$ // *1-move with prefix (suffix would place 7 next to 6)*
$\pi \leftarrow \pi{\cdot}\tau_p(2{,}7) = (\underline{11\ 12\ 13\ 14}\ \underline{1\ 4\ 5\ 6\ 2\ 3\ 8\ 9\ 10}\ 15\ 7)$ // *2-move with prefix*
$\pi \leftarrow \pi{\cdot}\tau_p(5{,}14) = (\underline{1}\ \underline{4\ 5\ 6}\ 2\ 3\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 7)$ // *1-move with prefix (suffix would put 7 next to 6)*
$\pi \leftarrow \pi{\cdot}\tau_p(2{,}5) = (4\ 5\ 6\ \underline{1\ 2\ 3\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15}\ \underline{7})$ // *1-move with suffix (prefix would put 6 next to 7)*
$\pi \leftarrow \pi{\cdot}\tau_s(4{,}15) = (\underline{4\ 5\ 6\ 7}\ \underline{1\ 2\ 3}\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$ // *2-move with prefix*
$\pi \leftarrow \pi{\cdot}\tau_p(5{,}8) = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$

Note that Lemma 9 directly yields the next lemma which, along with Lemma 1, directly yields Theorem 2.

Lemma 10. *For any unsigned permutation* π, `2-PST`$(\pi, n) \leq b_{ps}(\pi)$.

Theorem 2. SbPST *is 2-approximable.*

We point out that it is easy to implement `2-PST` to run in $O(n^2)$ time, since determining the right move to apply at each iteration takes $O(n)$ time, applying a transposition also takes $O(n)$ time, and the distance is linear in the number of breakpoints. Rusu [52], however, showed that it is possible to implement `2-PST` in $O(n \log n)$ time with a structure called *log-list*.

4.3 Sorting by Prefix and Suffix Reversals and Transpositions

The algorithm we propose for SbPSRT is called `2-PSRT` and it works as follows. While the permutation π has a upsrt-breakpoint, we try to apply a 2-move and, if this is not possible, then we apply a 1-move (which can be done with either of the four available rearrangements).

For this problem, due to the definition of upsrt-breakpoints (in which, in particular, $(1, n)$ and $(n, 1)$ are not upsrt-breakpoints), we define the unitary increment of a positive integer i as $inc(i, n) = (i \mod n) + 1$ and its unitary decrement as $dec(i, n) = ((i + n - 2) \mod n) + 1$, for $1 \leq i \leq n$.

To remove two upsrt-breakpoints we can use either a prefix or a suffix transposition, as Lemma 11 shows.

Lemma 11. *Let* $\pi \neq \iota_n$ *be any unsigned permutation. For* SbPSRT*, there exist at most eight* 2*-moves that can be applied to* π*.*

Proof. Suppose that $\tau_p(i, j)$ is a 2-move. We then must have $2 \leq i < j \leq n$. Also, $\pi{\cdot}\tau_p(i, j) = (\pi_i \ \ldots\ \pi_{j-1}\ \pi_1\ \ldots\ \pi_{i-1}\ \pi_j\ \ldots\ \pi_n)$, where $\pi_{j-1} = inc(\pi_1, n)$ or $\pi_{j-1} = dec(\pi_1, n)$ and $\pi_{i-1} = inc(\pi_j, n)$ or $\pi_{i-1} = dec(\pi_j, n)$.

Algorithm 2 A 2-approximation algorithm for SbPST.

2-PST(π, n)

Input: permutation π and its size n
Output: number of rearrangements used to sort π

$d \leftarrow 0$
while $\pi \neq \iota_n$ **do**
  Let $j_p = \pi^{-1}_{\pi_1 - 1} + 1$ and $i_p = \pi^{-1}_{\pi_{j_p} - 1} + 1$
  Let $i_s = \pi^{-1}_{\pi_n + 1}$ and $j_s = \pi^{-1}_{\pi_{i_s - 1} + 1}$
  if $2 \leq i_p < j_p \leq n$ **and** $2 \leq i_s < j_s \leq n$ **then** // If a 2-move is possible with a
    // prefix and a suffix transposition, apply the one that involves less elements
    if $j_p - 1 \leq n - i_s + 1$ **then**
      $\pi \leftarrow \pi \cdot \tau_p(i_p, j_p)$
    else
      $\pi \leftarrow \pi \cdot \tau_s(i_s, j_s)$
  else if $2 \leq i_p < j_p \leq n$ **then**
    $\pi \leftarrow \pi \cdot \tau_p(i_p, j_p)$
  else if $2 \leq i_s < j_s \leq n$ **then**
    $\pi \leftarrow \pi \cdot \tau_s(i_s, j_s)$
  else // A 2-move is not possible, so apply a 1-move
    Let i_p be the position of the last element of the first strip
    if $\pi_{i_p} = n$ **then**
      $j_p \leftarrow \pi^{-1}_{\pi_1 - 1} + 1$
    else
      $j_p \leftarrow \pi^{-1}_{\pi_{i_p} + 1}$
    Let j_s be the position of the first element of the last strip
    if $\pi_{j_s} = 1$ **then**
      $i_s \leftarrow \pi^{-1}_{\pi_n + 1} - 1$
    else
      $i_s \leftarrow \pi^{-1}_{\pi_{j_s} - 1}$
    // Choose the 1-move that involves less elements
    if $j_p - 1 \leq n - i_s$ **then**
      $\pi \leftarrow \pi \cdot \tau_p(i_p + 1, j_p)$
    else
      $\pi \leftarrow \pi \cdot \tau_s(i_s + 1, j_s)$
  $d \leftarrow d + 1$
return d

Now suppose that $\tau_s(i,j)$ is a 2-move. Again, $2 \leq i < j \leq n$ and $\pi \cdot \tau_s(i,j) = (\pi_1 \ldots \pi_{i-1}\ \pi_j \ldots \pi_n\ \pi_i \ldots \pi_{j-1})$, where $\pi_i = inc(\pi_n, n)$ or $\pi_i = dec(\pi_n, n)$ and $\pi_j = inc(\pi_{i-1}, n)$ or $\pi_j = dec(\pi_{i-1}, n)$. □

It is not always possible to have a 2-move in a permutation that contains upsrt-breakpoints, but a 1-move always appies, as Lemma 12 shows. Also, since for any permutation there may exist several 1-moves, we break ties by performing the one that affects the least number of elements. Algorithm 3 shows `2-PSRT`.

Lemma 12. *Let π be any unsigned permutation such that $b_{upsrt}(\pi) \geq 1$. For* SbPSRT*, it is always possible to apply a 1-move on π.*

Proof. If π_1 is in a decreasing strip or if it is a singleton, then let $\pi_j = inc(\pi_1, n)$. Now, if π_j is in an increasing strip or if it is a singleton, then the prefix reversal $\rho_p(j-1)$ is a 1-move; otherwise, let π_i be the first element of the strip that contains π_j. In this case, the prefix transposition $\tau_p(i, j+1)$ is a 1-move.

However, if π_1 is in an increasing strip, then let $\pi_j = dec(\pi_1, n)$. Now, if π_j is in a decreasing strip or if it is a singleton, then the prefix reversal $\rho_p(j-1)$ is a 1-move; otherwise, let π_i be the first element of the strip that contains π_j. In this case, the prefix transposition $\tau_p(i, j+1)$ is a 1-move.

Similarly, if π_n is in a decreasing strip or if it is a singleton, then let $\pi_i = dec(\pi_n, n)$. If π_i is in an increasing strip or if it is a singleton, then the suffix reversal $\rho_s(i+1)$ is a 1-move; otherwise, let π_j be the last element of the strip that contains π_i. In this case, the suffix transposition $\tau_s(i, j+1)$ is a 1-move.

Finally, if π_n is in an increasing strip, then let $\pi_i = inc(\pi_n, n)$. Now, if π_i is in a decreasing strip or if it is a singleton, then the suffix reversal $\rho_s(i+1)$ is a 1-move; otherwise, let π_j be the last element of the strip that contains π_i. In this case, the suffix transposition $\tau_s(i, j+1)$ is a 1-move. □

Example 11. *The following example shows the execution of* `2-PSRT` *over* $\pi = (9\ 6\ 2\ 14\ 10\ 15\ 7\ 12\ 4\ 11\ 3\ 8\ 13\ 1\ 5)$*:*

$\pi = (\underline{9\ 6\ 2\ 14\ 10\ 15\ 7\ 12}\ \underline{4\ 11\ 3\ 8}\ 13\ 1\ 5)$ // *2-move with prefix (there is a 2-move with suffix)*
$\pi \leftarrow \pi{\cdot}\tau_p(9,13) = (4\ 11\ 3\ 8\ 9\ \underline{6\ 2\ 14}\ \underline{10\ 15\ 7\ 12}\ \underline{13\ 1\ 5})$ // *2-move with suffix*
$\pi \leftarrow \pi{\cdot}\tau_s(6,9) = (\underline{4\ 11}\ 3\ 8\ 9\ 10\ 15\ 7\ 12\ 13\ 1\ 5\ 6\ 2\ 14)$ // *1-move with prefix reversal (least # of elements)*
$\pi \leftarrow \pi{\cdot}\rho_p(2) = (11\ 4\ 3\ 8\ 9\ 10\ 15\ 7\ 12\ 13\ \underline{1\ 5\ 6\ 2\ 14})$ // *1-move with suffix reversal*
$\pi \leftarrow \pi{\cdot}\rho_s(11) = (\underline{11}\ \underline{4\ 3\ 8\ 9\ 10}\ 15\ 7\ 12\ 13\ \underline{14\ 2\ 6\ 5\ 1})$ // *1-move with prefix transposition*
$\pi \leftarrow \pi{\cdot}\tau_p(2,7) = (\underline{4\ 3\ 8\ 9\ 10\ 11\ 15}\ \underline{7\ 12\ 13\ 14\ 2\ 6\ 5}\ 1)$ // *2-move with prefix (places n close to 1)*
$\pi \leftarrow \pi{\cdot}\tau_p(8,15) = (7\ 12\ 13\ 14\ \underline{2\ 6\ 5\ 4\ 3\ 8\ 9\ 10\ 11\ 15\ 1})$ // *2-move with suffix*
$\pi \leftarrow \pi{\cdot}\tau_s(5,14) = (7\ \underline{12\ 13\ 14\ 15\ 1\ 2}\ \underline{6\ 5\ 4\ 3\ 8\ 9\ 10\ 11})$ // *2-move with suffix*
$\pi \leftarrow \pi{\cdot}\tau_s(2,8) = (\underline{7\ 6\ 5\ 4\ 3}\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 1\ 2)$ // *1-move with prefix reversal*
$\pi \leftarrow \pi{\cdot}\rho_p(5) = (\underline{3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15}\ \underline{1\ 2})$ // $\pi = \varphi^a_{k,n}$ *for* $k = 3$
$\pi \leftarrow \pi{\cdot}\tau_p(14,16) = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$

Lemma 13 and Theorem 3 show how we can guarantee an approximation factor of $2 + 4/b_{upsrt}(\pi)$ for `2-PSRT`.

Lemma 13. *For any unsigned permutation π, `2-PSRT`$(\pi, n) \leq b_{upsrt}(\pi) + 2$.*

Proof. Our algorithm always removes at least one upsrt-breakpoint from π with one rearrangement, which is possible according to Lemma 12. When π has no upsrt-breakpoints, by the definition of breakpoints we know that π must be of one of the following four forms:

Algorithm 3 A $(2 + 4/b_{upsrt}(\pi))$-approximation algorithm for SbPSRT.

2-PSRT(π, n)

Input: permutation π and its size n
Output: number of rearrangements used to sort π

$d \leftarrow 0$
while $\pi \neq \iota_n$ **do**
  if $\pi = \eta_n$ **then**
    $\pi \leftarrow \pi \cdot \rho_p(n)$
    $d \leftarrow d + 1$
  else if $\tau_p(i, j)$ is a 2-move where $((\pi_{j-1} = dec(\pi_1, n)$ and $\pi_{i-1} = dec(\pi_j, n))$ **or** $(\pi_{j-1} = dec(\pi_1, n)$ and $\pi_{i-1} = inc(\pi_j, n))$ **or** $(\pi_{j-1} = inc(\pi_1, n)$ and $\pi_{i-1} = dec(\pi_j, n))$ **or** $(\pi_{j-1} = inc(\pi_1, n)$ and $\pi_{i-1} = inc(\pi_j, n)))$ **then**
    $\pi \leftarrow \pi \cdot \tau_p(i, j)$
    $d \leftarrow d + 1$
  else if $\tau_s(i, j)$ is a 2-move where $((\pi_i = dec(\pi_n, n)$ and $\pi_j = dec(\pi_{i-1}, n))$ **or** $(\pi_i = dec(\pi_n, n)$ and $\pi_j = inc(\pi_{i-1}, n))$ **or** $(\pi_i = inc(\pi_n, n)$ and $\pi_j = dec(\pi_{i-1}, n))$ **or** $(\pi_i = inc(\pi_n, n)$ and $\pi_j = inc(\pi_{i-1}, n)))$ **then**
    $\pi \leftarrow \pi \cdot \tau_s(i, j)$
    $d \leftarrow d + 1$
  else // Otherwise, apply a 1-move
    // First, find all possible positions to apply a 1-move with prefix
    If π_1 is in a decreasing strip or it is a singleton, let $j_1 = \pi^{-1}_{inc(\pi_1,n)}$. If π_{j_1} is in a decreasing strip, then let i_1 be the first element of the strip that contains π_{j_1}, else let $j_2 = j_1 - 1$
    If π_1 is in an increasing strip, let $j_1 = \pi^{-1}_{dec(\pi_1,n)}$. If π_{j_1} is in an increasing strip, then let i_1 be the first element of the strip that contains π_{j_1}, else let $j_2 = j_1 - 1$
    // Now, find all possible positions to apply a 1-move with suffix
    If π_n is in a decreasing strip or it is a singleton, let $i_3 = \pi^{-1}_{dec(\pi_n,n)}$. If π_{i_3} is in a decreasing strip, then let j_3 be the last element of the strip that contains π_{i_3}, else let $i_4 = i_3 + 1$
    If π_n is in an increasing strip, let $i_3 = \pi^{-1}_{inc(\pi_n,n)}$. If π_{i_3} is in an increasing strip, then let j_3 be the last element of the strip that contains π_{i_3}, else let $i_4 = i_3 + 1$
    Let λ be the rearrangement that involves the least number of elements among $\tau_p(i_1, j_1 + 1)$, $\rho_p(j_2)$, $\tau_s(i_3, j_3 + 1)$, and $\rho_s(i_4)$ (for i_x and j_y that are defined)
    if λ exists **then**
      $\pi \leftarrow \pi \cdot \lambda$
      $d \leftarrow d + 1$
    else if $\pi = \varphi^a_{k,n}$ for some $2 \leq k \leq n$ **then**
      $\pi \leftarrow \pi \cdot \tau_p(\pi_n^{-1} + 1, n + 1)$
      $d \leftarrow d + 1$
    else // $\pi = \varphi^d_{k,n}$ for some $2 \leq k \leq n$
      $\pi \leftarrow \pi \cdot \tau_p(\pi_n^{-1}, n + 1) \cdot \rho_p(n)$
      $d \leftarrow d + 2$
return d

ι_n, η_n, $\varphi^a_{k,n} = (k\ k+1\ k+2\ \ldots\ n\ 1\ 2\ 3\ \ldots\ k-1)$, or $\varphi^d_{k,n} = (k-1\ k-2\ k-3\ \ldots\ 1\ n\ n-1\ n-2\ \ldots\ k)$, for $2 \leq k \leq n$ in both cases.

It is easy to note that ι_n is sorted ($d_{psrt}(\iota_n) = 0$), η_n is one reversal distant from being sorted ($d_{psrt}(\eta_n) = 1$), $\varphi^a_{k,n}$ is one transposition distant from being sorted ($d_{psrt}(\varphi^a_{k,n}) = 1$), and $\varphi^d_{k,n}$ is one transposition and one reversal distant from being sorted ($d_{psrt}(\varphi^d_{k,n}) = 2$). Therefore, at most $b_{upsrt}(\pi) + 2$ rearrangements can sort any permutation. □

Theorem 3. SBPSRT *is* $(2 + 4/b_{upsrt}(\pi))$*-approximable.*

Proof. First note that `2-PSRT`, presented in Algorithm 3, can also be easily implemented to run in $O(n^2)$ time, but the structure of log-lists presented by Rusu [52] can improve this to $O(n \log n)$ time.

Now, from Lemmas 1 and 13, we have that the theoretical approximation factor for `2-PSRT` is $(b_{upsrt}(\pi) + 2)/(b_{upsrt}(\pi)/2) = 2 + 4/b_{upsrt}(\pi)$. □

4.4 Sorting by Signed Prefix and Suffix Reversals

We also propose an algorithm specific for SBPSR̄, called `2-PSR̄`, which also greedily removes breakpoints. The idea of `2-PSR̄` is somewhat similar to the idea of `2-PSR`. First, we try to find a 1-move; if this is not possible, then we try to remove one ps-breakpoint with a 0-move followed by a 1-move; if this is not possible either, then the permutation has a special format and we can use a specific sequence to sort it with few prefix and suffix reversals. Lemmas 14 and 15 analyse how to remove one ps-breakpoint while Lemmas 16 and 17 show the special format and the specific sequence to sort it, respectively.

Lemma 14. *Let* $\pi \neq \iota_n$ *be any signed permutation. For* SBPSR̄*, there exists at most two* 1*-moves that can be applied to* π*.*

Proof. If $\pi_j = -\pi_1 + 1$ for some $1 < j \leq n$, then $\bar{\rho}_p(j-1)$ is a 1-move. Likewise, if $\pi_i = -\pi_n - 1$ for some $1 \leq i < n$, then $\bar{\rho}_s(i+1)$ is a 1-move. No other 1-move is possible, because the first or the last elements must be involved. □

Lemma 15. *Let* $\pi \neq \iota_n$ *be any signed permutation. For* SBPSR̄*, there is a sequence of at most two prefix or suffix reversals that removes one ps-breakpoint if there are* π_i *and* π_j *such that at least one of the following cases happens:*

1. $\pi_j = -\pi_i - 1$ *for* $1 \leq i < j \leq n$*;*
2. $\pi_j = \pi_i + 1$ *for* $0 \leq i + 1 < j \leq n$*;*
3. $\pi_i = -\pi_j + 1$ *for* $1 \leq i < j \leq n$*.*

Proof. If π_i and π_j exist such that

1. $\pi_j = -\pi_i - 1$ for $1 \leq i < j \leq n$, then we have $\pi = (\ldots\ldots\ k\ \ldots\ldots\ -(k+1)\ \ldots\ldots)$ and $\bar{\rho}_p(j) \cdot \bar{\rho}_p(j-i)$ is a 0-move followed by a 1-move;
2. $\pi_j = \pi_i + 1$ for $0 \leq i + 1 < j \leq n$, then we have $\pi = (\ldots\ldots\ k\ \ldots\ldots\ k+1\ \ldots\ldots)$ and $\bar{\rho}_p(i) \cdot \bar{\rho}_p(j-1)$ is a 0-move followed by a 1-move;

3. $\pi_i = -\pi_j + 1$ for $1 \leq i < j \leq n$, then we have $\pi = (\ldots\ldots\ k\ \ldots\ldots\ -(k-1)\ \ldots\ldots)$ and $\bar{\rho}_s(i) \cdot \bar{\rho}_s(n+1-(j-i))$ is a 0-move followed by a 1-move.

□

Lemma 16. *For* SBPSR̄*, if π is any signed permutation for which it is not possible to find a* 1*-move, or a* 0*-move followed by a* 1*-move, then π is of one of the three following forms:*

1. $\bar{\eta_n} = (-n\ -(n-1)\ -(n-2)\ \ldots\ -1)$;
2. $\delta^{sa}_{b+1} = (\underbrace{p_b{+}1\ p_b{+}2\ \ldots\ n}_{\ell_{b+1}}\ \ \underbrace{p_{b-1}{+}1\ p_{b-1}{+}2\ \ldots\ p_b}_{\ell_b}\ \ \ldots\ldots\ \ \underbrace{1\ 2\ \ldots\ p_1}_{\ell_1})$;
3. $\delta^{sd}_{b+1} = (\underbrace{-p_1\ -(p_1{-}1)\ \ldots\ -1}_{\ell_1}\ \ \underbrace{-p_2\ -(p_2{-}1)\ \ldots\ -(p_1{+}1)}_{\ell_2}\ \ \ldots\ldots\ldots\ldots\ \underbrace{-n\ -(n{-}1)\ \ldots\ -(p_b{+}1)}_{\ell_{b+1}})$,

where $b = b_{ps}(\pi) \geq 1$ and $\ell_i \geq 1$ for all $1 \leq i \leq b_{ps}(\pi) + 1$.

Proof. It is easy to see that when $\pi = \bar{\eta_n}$ the greedy part of the algorithm cannot turn it into the identity, since a reversal $\bar{\rho}_p(n)$ or $\bar{\rho}_s(1)$ is necessary and neither removes a ps-breakpoint, because $b_{ps}(\bar{\eta_n})$ is already zero.

Let $\pi_i = k$ be any element of π. If $\pi_j = -(k{+}1)$ with $i < j$, then $\bar{\rho}_p(j) \cdot \bar{\rho}_p(j-i)$ removes one ps-breakpoint. If $j < i$, then $\bar{\rho}_s(j) \cdot \bar{\rho}_s(n+1-(i-j))$ removes one ps-breakpoint. Something similar happens when $-(k-1)$ exists in π. Therefore, the elements of π must all have the same sign.

Suppose that π has only positive elements. If $b_{ps}(\pi) = 1$, it is trivial to see that π must be of the form δ^{sa}_2, which contains $s = 2$ strips. Now assume that every permutation with $s - 1 \geq 2$ positive strips for which it is not possible to find a 1-move or a 0-move followed by a 1-move is of the form δ^{sa}_{s-1}. Note that $s = b + 1$.

Let π be a permutation with s positive strips for which it is not possible to find a 1-move or a 0-move followed by a 1-move and let π_w, for $1 < w \leq n$, be the first element of the last strip of π. Note that we must have $\pi_w = 1$, otherwise we would have a $\pi_i = \pi_w - 1$ for some $i + 1 < w$, which is a contradiction, since $\bar{\rho}_p(i) \cdot \bar{\rho}_p(w-1)$ could remove a ps-breakpoint.

Let π' be a permutation with $w - 1$ elements built from π such that $\pi'_i = \pi_i - \pi_n$ for all $1 \leq i < w$ (in other words, π' is π without the last strip and relabeled accordingly). It is easy to see that π' has $s - 1$ positive strips and one can see that if it would be possible to remove one ps-breakpoint from π', then it would also be possible to remove it from π. By induction hypothesis, π' is of the form δ^{sa}_{s-1}. Since π' is π relabeled without the last strip, which has the element 1, it follows that π is also of the form δ^{sa}_s.

When π only contains negative elements, a similar proof applies to show that π is of the form δ^{sd}_{b+1}. □

Lemma 17. *Let π be one of the signed permutations described in Lemma 16. If $\pi = \bar{\eta_n}$, then one signed prefix reversal $\bar{\rho}_p(n)$ sorts it. Otherwise, at most $b_{ps}(\pi) + 2$ prefix and suffix reversals sort it.*

Proof. Let $b = b_{ps}(\pi)$ for simplicity. If $\pi = \delta^{sa}_{b+1}$ and b is an even number, then the $b+1$ reversals

$$\bar{\rho}_p(n-\ell_1)\cdot\bar{\rho}_s(\ell_2+1)\cdot\bar{\rho}_p(n-\ell_3)\cdot\bar{\rho}_s(\ell_4+1)\cdot\ldots\cdot\bar{\rho}_p(n-\ell_{b-1})\cdot\bar{\rho}_s(\ell_b+1)\cdot\bar{\rho}_p(n-\ell_{b+1})$$

sort π, as we show next.

Let π^k, $1 \le k \le b/2$, be the permutation we obtain after applying the first $2k$ reversals $\bar{\rho}_p(n-\ell_1)\cdot\bar{\rho}_s(\ell_2+1)\cdot\ldots\cdot\bar{\rho}_p(n-\ell_{2k-1})\cdot\bar{\rho}_s(\ell_{2k}+1)$ of the sequence given above. We will show by induction on k that π^k is of the form $(-p_{2k}\ldots-(p_{2k-1}+1)\ -p_{2k-1}\ldots-(p_{2k-2}+1)$ $\ldots\ldots-p_1\ldots-1\ p_b+1\ldots n\ \ldots\ldots\ p_{2k+1}+1\ldots p_{2k+2}\ p_{2k}+1\ldots p_{2k+1})$, where $\langle -p_{2k}\ldots-1\rangle$ has size $\ell_{2k}+\ell_{2k-1}+\ldots+\ell_1$, $\langle p_b+1\ldots n\rangle$ has size ℓ_{b+1}, $\langle p_{2k+1}+1\ldots p_{2k+2}\rangle$ has size ℓ_{2k+2}, and $\langle p_{2k}+1\ldots p_{2k+1}\rangle$ has size ℓ_{2k+1}.

It is easy to see that π^k has this form when $k=1$. Now, assume that π^{k-1}, for $k-1 \ge 1$, is of this form. Since $\pi^k = \pi^{k-1}\cdot\bar{\rho}_p(n-\ell_{2k-1})\cdot\bar{\rho}_s(\ell_{2k}+1)$, the result follows.

Now, when $k = b/2$, $\pi^{b/2} = (-p_b\ldots-(p_{b-1}+1)\ \ldots\ -p_1\ \ldots\ -1\ p_b+1\ \ldots\ n)$ and one reversal, namely the last reversal of the sequence, $\bar{\rho}_p(n-\ell_{b+1})$, sorts $\pi^{b/2}$.

If $\pi = \delta^{sd}_{b+1}$ and b is an even number, then one can apply $\bar{\rho}_p(n)$ to transform it into δ^{sa}_{b+1} followed by the $b+1$ reversals given above.

If $\pi = \delta^{sd}_{b+1}$ and b is an odd number, then the $b+1$ reversals

$$\bar{\rho}_s(\ell_1+1)\cdot\bar{\rho}_p(n-\ell_2)\cdot\bar{\rho}_s(\ell_3+1)\cdot\bar{\rho}_p(n-\ell_4)\cdot\ldots\cdot\bar{\rho}_s(\ell_b+1)\cdot\bar{\rho}_p(n-\ell_{b+1})$$

sort π. This can be shown by a similar induction as the one done above.

If $\pi = \delta^{sa}_{b+1}$ and b is odd, then one can apply $\bar{\rho}_p(n)$ to transform it into δ^{sd}_{b+1} followed by the reversals above. □

Since it is possible to have both prefix reversals and suffix reversals for the 1-move or for the 0-move followed by the 1-move, we decided to interpolate the choices, as shown in Algorithm 4, which presents 2-PSR̄.

Example 12. *The following example shows the execution of* 2-PSR̄ *over* π = (–9 –7 –8 –15 4 –12 6 3 –14 –11 –5 –13 10 2 –1)*:*

π	= (–9 –7 –8 –15 4 –12 6 3 –14 –11 –5 –13 10 2 –1)	// *1-move with prefix*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(12)$	= (13 5 11 14 –3 –6 12 –4 15 8 7 9 10 2 –1)	// *0-move (π has $\pi_j = \pi_i + 1$)*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(1)$	= (–13 5 11 14 –3 –6 12 –4 15 8 7 9 10 2 –1)	// *1-move with prefix (following the 0-move)*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(3)$	= (–11 –5 13 14 –3 –6 12 –4 15 8 7 9 10 2 –1)	// *1-move with prefix*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(6)$	= (6 3 –14 –13 5 11 12 –4 15 8 7 9 10 2 –1)	// *0-move (π has $\pi_j = \pi_i + 1$)*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(1)$	= (–6 3 –14 –13 5 11 12 –4 15 8 7 9 10 2 –1)	// *1-move with prefix (following the 0-move)*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(10)$	= (–8 –15 4 –12 –11 –5 13 14 –3 6 7 9 10 2 –1)	// *1-move with prefix*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(11)$	= (–7 –6 3 –14 –13 5 11 12 –4 15 8 9 10 2 –1)	// *1-move with prefix*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(10)$	= (–15 4 –12 –11 –5 13 14 –3 6 7 8 9 10 2 –1)	// *0-move (π has $\pi_j = -\pi_i - 1$)*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(7)$	= (–14 –13 5 11 12 –4 15 –3 6 7 8 9 10 2 –1)	// *1-move with prefix (following the 0-move)*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(6)$	= (4 –12 –11 –5 13 14 15 –3 6 7 8 9 10 2 –1)	// *1-move with prefix*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(7)$	= (–15 –14 –13 5 11 12 –4 –3 6 7 8 9 10 2 –1)	// *0-move (π has $\pi_j = -\pi_i - 1$)*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(6)$	= (–12 –11 –5 13 14 15 –4 –3 6 7 8 9 10 2 –1)	// *1-move with prefix (following the 0-move)*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(3)$	= (5 11 12 13 14 15 –4 –3 6 7 8 9 10 2 –1)	// *1-move with prefix*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(6)$	= (–15 –14 –13 –12 –11 –5 –4 –3 6 7 8 9 10 2 –1)	// *0-move (π has $\pi_j = -\pi_i - 1$)*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(13)$	= (–10 –9 –8 –7 –6 3 4 5 11 12 13 14 15 2 –1)	// *1-move with prefix (following the 0-move)*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(8)$	= (–5 –4 –3 6 7 8 9 10 11 12 13 14 15 2 –1)	// *1-move with prefix*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(3)$	= (3 4 5 6 7 8 9 10 11 12 13 14 15 2 –1)	// *0-move (π has $\pi_i = -\pi_j + 1$)*
$\pi \leftarrow \pi\cdot\bar{\rho}_s(14)$	= (3 4 5 6 7 8 9 10 11 12 13 14 15 1 –2)	// *1-move with suffix (following the 0-move)*
$\pi \leftarrow \pi\cdot\bar{\rho}_s(15)$	= (3 4 5 6 7 8 9 10 11 12 13 14 15 1 2)	// *$\pi = \delta^{sa}_{b+1}$ for $b = 1$ (odd, so apply $\bar{\rho}_p(n)$)*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(15)$	= (–2 –1 –15 –14 –13 –12 –11 –10 –9 –8 –7 –6 –5 –4 –3)	// *apply $\bar{\rho}_s(\ell_1+1)$*
$\pi \leftarrow \pi\cdot\bar{\rho}_s(3)$	= (–2 –1 3 4 5 6 7 8 9 10 11 12 13 14 15)	// *apply $\bar{\rho}_p(n-\ell_2)$*
$\pi \leftarrow \pi\cdot\bar{\rho}_p(2)$	= (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)	

Lemma 18 and Theorem 4 show that 2-PS$\bar{\text{R}}$ is a 2-approximation.

Lemma 18. *For any signed permutation π, 2-PS$\bar{\text{R}}$$(\pi, n) \leq 2b_{ps}(\pi) + 1$.*

Proof. Starting from π, our algorithm iterates by doing a greedy removal of breakpoints until this is not possible anymore. At worse, it uses two prefix or suffix reversals to remove one breakpoint, as Lemma 15 shows. Suppose it does this until π' is reached. If $b_{ps}(\pi') = 0$, then either $\pi' = \iota_n$ or $\pi' = \bar{\eta_n}$. Since $d_{ps\bar{r}}(\iota_n) = 0$ and $d_{ps\bar{r}}(\bar{\eta_n}) = 1$, we have 2-PS$\bar{\text{R}}$$(\pi, n) \leq 2b_{ps}(\pi) + 1$.

If $b_{ps}(\pi') \geq 1$, then 2-PS$\bar{\text{R}}$$(\pi, n) \leq 2(b_{ps}(\pi) - b_{ps}(\pi')) +$ 2-PS$\bar{\text{R}}$$(\pi', n)$. According to Lemma 17, 2-PS$\bar{\text{R}}$$(\pi', n) \leq b_{ps}(\pi') + 2$. Therefore, 2-PS$\bar{\text{R}}$$(\pi, n) \leq 2b_{ps}(\pi) - b_{ps}(\pi') + 2 \leq 2b_{ps}(\pi) + 1$. □

Theorem 4. SbPS$\bar{\text{R}}$ *is 2-approximable.*

Proof. First note that it is easy to implement 2-PS$\bar{\text{R}}$ to run in $O(n^2)$ time.

Now, if the input permutation π is such that $b_{ps}(\pi) = 0$, then either $\pi = \iota_n$ or $\pi = \bar{\eta_n}$. In any case, 2-PS$\bar{\text{R}}$ is optimal. Therefore, suppose $b_{ps}(\pi) \geq 1$.

Suppose that it is possible to apply a 1-move on π, generating a permutation π'. It is easy to see that 2-PS$\bar{\text{R}}$$(\pi) \leq 1 +$ 2-PS$\bar{\text{R}}$$(\pi')$ and $b_{ps}(\pi') = b_{ps}(\pi) - 1$. By Lemma 18, we know that 2-PS$\bar{\text{R}}$$(\pi') \leq 2b_{ps}(\pi') + 1$. Therefore,

$$\text{2-PS}\bar{\text{R}}(\pi) \leq 1 + \text{2-PS}\bar{\text{R}}(\pi') \leq 1 + 2(b_{ps}(\pi) - 1) + 1 = 2b_{ps}(\pi).$$

By Lemma 2, we know that $d_{ps\bar{r}}(\pi) \geq b_{ps}(\pi)$, and therefore 2-PS$\bar{\text{R}}$ is a 2-approximation algorithm in that case.

Now suppose that it is not possible to apply a 1-move on π. This means that the lower bound given in Lemma 2 is not tight, which implies that $d_{ps\bar{r}}(\pi) \geq b_{ps}(\pi)+1$. Using Lemma 18, we have that the theoretical approximation factor of 2-PS$\bar{\text{R}}$ is

$$\frac{2b_{ps}(\pi) + 1}{b_{ps}(\pi) + 1} \leq \frac{2b_{ps}(\pi) + 2}{b_{ps}(\pi) + 1} = 2.$$

□

4.5 Sorting by Signed Prefix Reversals and Transpositions and Sorting by Signed Prefix and Suffix Reversals and Transpositions

The two algorithms we developed for SbP$\bar{\text{R}}$T and SbPS$\bar{\text{R}}$T, called 2-P$\bar{\text{R}}$T and 2-PS$\bar{\text{R}}$T, are very similar to 2-PRT and 2-PSRT, respectively. For 2-PS$\bar{\text{R}}$T, we also use $inc(i, n)$ and $dec(i, n)$ as defined in Section 4.3 to increment and decrement positive integers, but we need to define such functions for negative integers. The unitary increment for a negative integer i is $inc(i, n) = -(((-i + n - 2) \mod n) + 1)$ while its unitary decrement is $dec(i, n) = -((-i \mod n) + 1)$, for $-n \leq i \leq -1$.

Algorithm 4 A 2-approximation algorithm for SbPS$\bar{\text{R}}$.

2-PS$\bar{\text{R}}$(π, n)

Input: permutation π and its size n
Output: number of rearrangements used to sort π

$d \leftarrow 0$
while $\pi \neq \iota_n$ **do**
  if $\pi_j = -\pi_i + 1$ for $1 < j \leq n$ **then**
    $\pi \leftarrow \pi \cdot \bar{\rho}_p(j-1)$
    $d \leftarrow d + 1$
  else if $\pi_i = -\pi_n - 1$ for $1 \leq i < n$ **then**
    $\pi \leftarrow \pi \cdot \bar{\rho}_s(i+1)$
    $d \leftarrow d + 1$
  else if $\pi_j = -\pi_i - 1$ for $1 \leq i < j \leq n$ **then**
    $\pi \leftarrow \pi \cdot \bar{\rho}_p(j) \cdot \bar{\rho}_p(j-i)$
    $d \leftarrow d + 2$
  else if $\pi_j = \pi_i + 1$ for $0 \leq i+1 < j \leq n$ **then**
    $\pi \leftarrow \pi \cdot \bar{\rho}_p(i) \cdot \bar{\rho}_p(j-1)$
    $d \leftarrow d + 2$
  else if $\pi_i = -\pi_j + 1$ for $1 \leq i < j \leq n$ **then**
    $\pi \leftarrow \pi \cdot \bar{\rho}_s(i) \cdot \bar{\rho}_s(n+1-(j-i))$
    $d \leftarrow d + 2$
  else if $\pi = \bar{\eta_n}$ **then**
    $\pi \leftarrow \pi \cdot \bar{\rho}_s(1)$
    $d \leftarrow d + 1$
  else // $\pi = \delta^{sa}_{b+1}$ or $\pi = \delta^{sd}_{b+1}$ (Lemma 16)
    $b \leftarrow b_{ps}(\pi)$
    if $b \bmod 2 \equiv 1$ **then**
      if $\pi = (p_b{+}1 \ \ldots \ n \ p_{b-1}{+}1 \ \ldots \ p_b \ \ldots\ldots \ 1 \ \ldots \ p_1) = \delta^{sa}_{b+1}$ **then**
        $\pi \leftarrow \pi \cdot \bar{\rho}_p(\mathtt{n})$
        $d \leftarrow d + 1$
      Let ℓ_{b+2-i} be the size of the ith strip of π
      // Apply the rearrangements from Lemma 17 for δ^{sd}_{b+1} when $b_{ps}(\pi)$ is odd
      $\pi \leftarrow \pi \cdot \bar{\rho}_p(n-\ell_1) \cdot \bar{\rho}_s(\ell_2+1) \cdot \cdots \cdot \bar{\rho}_p(n-\ell_{b-1}) \cdot \bar{\rho}_s(\ell_b+1) \cdot \bar{\rho}_p(n-\ell_{b+1})$
    else
      if $\pi = (-p_1 \ldots -1 \ -p_2 \ldots -(p_1{+}1) \ldots\ldots -n \ldots -(p_b{+}1)) = \delta^{sd}_{b+1}$ **then**
        $\pi \leftarrow \pi \cdot \bar{\rho}_p(n)$
        $d \leftarrow d + 1$
      Let ℓ_i be the size of the ith strip of π
      // Apply the rearrangements from Lemma 17 for δ^{sa}_{b+1} when $b_{ps}(\pi)$ is even
      $\pi \leftarrow \pi \cdot \bar{\rho}_s(\ell_1+1) \cdot \bar{\rho}_p(n-\ell_2) \cdot \cdots \cdot \bar{\rho}_s(\ell_b+1) \cdot \bar{\rho}_p(n-\ell_{b+1})$
    $d \leftarrow d + b + 1$
return d

First we discuss algorithm 2-PR̄T, which we propose for SbPR̄T. It works as follows while the permutation is not sorted: if $\pi_1 = 1$, we move the first strip to the end of the permutation with a prefix transposition; otherwise, we try to apply a 2-move in the form of a prefix transposition $\tau_p(i, j)$ where $\pi_i \neq 1$, as shown in Lemma 19, and, if such 2-move is not possible, we apply a 1-move, which is always possible, as shown by Lemma 20. We show 2-PR̄T in Algorithm 5.

Example 13. *The following example shows the execution of* 2-PR̄T *over* $\pi = (-9\ -7\ -8\ -15\ 4\ -12\ 6\ 3\ -14\ -11\ -5\ -13\ 10\ 2\ -1)$*:*

$\pi = (-9\ -7\ -8\ -15\ 4\ -12\ 6\ 3\ -14\ -11\ -5\ -13\ 10\ 2\ -1)$ // $\pi_i = -\pi_1 + 1$ *exists (so pref. rev.)*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_p(12) = (13\ 5\ 11\ 14\ -3\ -6\ 12\ -4\ 15\ 8\ 7\ 9\ 10\ 2\ -1)$ // $\pi_{j-1} = \pi_1 - 1$ *exists (so pref. transp.)*
$\pi \leftarrow \pi{\cdot}\tau_p(2,8) = (5\ 11\ 14\ -3\ -6\ 12\ 13\ -4\ 15\ 8\ 7\ 9\ 10\ 2\ -1)$ // *1-move with prefix reversal*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_p(7) = (-13\ -12\ 6\ 3\ -14\ -11\ -5\ -4\ 15\ 8\ 7\ 9\ 10\ 2\ -1)$ // *2-move*
$\pi \leftarrow \pi{\cdot}\tau_p(3,6) = (6\ 3\ -14\ -13\ -12\ -11\ -5\ -4\ 15\ 8\ 7\ 9\ 10\ 2\ -1)$ // *1-move with prefix reversal*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_p(6) = (11\ 12\ 13\ 14\ -3\ -6\ -5\ -4\ 15\ 8\ 7\ 9\ 10\ 2\ -1)$ // *1-move with prefix transposition*
$\pi \leftarrow \pi{\cdot}\tau_p(5,14) = (-3\ -6\ -5\ -4\ 15\ 8\ 7\ 9\ 10\ 11\ 12\ 13\ 14\ 2\ -1)$ // *1-move with prefix transposition*
$\pi \leftarrow \pi{\cdot}\tau_p(2,5) = (-6\ -5\ -4\ -3\ 15\ 8\ 7\ 9\ 10\ 11\ 12\ 13\ 14\ 2\ -1)$ // *1-move with prefix reversal*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_p(6) = (-8\ -15\ 3\ 4\ 5\ 6\ 7\ 9\ 10\ 11\ 12\ 13\ 14\ 2\ -1)$ // *1-move with prefix reversal*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_p(7) = (-7\ -6\ -5\ -4\ -3\ 15\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 2\ -1)$ // *1-move with prefix reversal*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_p(6) = (-15\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 2\ -1)$ // *1-move with prefix reversal*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_p(15) = (1\ -2\ -14\ -13\ -12\ -11\ -10\ -9\ -8\ -7\ -6\ -5\ -4\ -3\ 15)$ // *π starts at 1, so send first strip to end*
$\pi \leftarrow \pi{\cdot}\tau_p(2,16) = (-2\ -14\ -13\ -12\ -11\ -10\ -9\ -8\ -7\ -6\ -5\ -4\ -3\ 15\ 1)$ // *1-move with prefix transposition*
$\pi \leftarrow \pi{\cdot}\tau_p(2,14) = (-14\ -13\ -12\ -11\ -10\ -9\ -8\ -7\ -6\ -5\ -4\ -3\ -2\ 15\ 1)$ // *1-move with prefix reversal*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_p(13) = (2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 1)$ // *only time n and 1 are separated*
$\pi \leftarrow \pi{\cdot}\tau_p(15,16) = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$

Algorithm 5 A $(2 + 4/b_p(\pi))$-approximation algorithm for SbPR̄T.

2-PR̄T(π, n)

Input: permutation π and its size n
Output: number of rearrangements used to sort π

```
1  d ← 0
2  while π ≠ ι_n do
3      if π_1 = 1 then
4          Let π_i be the last element of the first strip of π
5          π ← π · τ_p(i + 1, n + 1)
6      else if π_{j−1} = π_1 − 1 and π_{i−1} = π_j − 1 and 2 ≤ i < j and π_i ≠ 1 then
7          π ← π · τ_p(i, j)
8      else if π_{j−1} = π_1 − 1 then
9          Let π_i be the last element of the first strip
10         π ← π · τ_p(i + 1, j)
11     else
12         Let π_i = −π_1 + 1
13         π ← π · ρ̄_p(i − 1)
14     d ← d + 1
15 return d
```

Lemmas 19 to 23 will be used by Lemma 24 and Theorem 5 to show how this algorithm guarantees an approximation factor of $2 + 4/b_p(\pi)$.

Lemma 19. *Let $\pi \neq \iota_n$ be any signed permutation with $\pi_1 \neq 1$. For* SbPR̄T*, there exists at most one 2-move that can be applied to π.*

Proof. Suppose that $\tau_p(i, j)$ is a 2-move. Note that we then must have $2 \leq i < j \leq n+1$. Also, $\pi \cdot \tau_p(i, j) = (\pi_i \ldots \pi_{j-1}\ \pi_1 \ldots \pi_{i-1}\ \pi_j \ldots \pi_n)$, where $\pi_{i-1} = \pi_j - 1 \neq \pi_i - 1$ and π_{j-1}

$= \pi_1 - 1 \neq \pi_j - 1$. It is easy to see that π_1 uniquely determines j and that j uniquely determines i. Note that $\pi_1 - 1$ or $\pi_j - 1$ do not necessarily exist in π. □

Lemma 20. *Let $\pi \neq \iota_n$ be any signed permutation with $\pi_1 \neq 1$. For* SbP$\bar{\text{R}}$T*, it is always possible to apply a 1-move on π.*

Proof. Consider the first element $\pi_1 = k$ of the permutation and let π_i be the last element of the first strip of π. We must have that either $k-1$ or $-(k-1)$ exists in π at some position $j > 1$, since $k \neq 1$. If $k-1$ exists, then either it is a singleton or it is located at the end of some strip. In both cases, a transposition $\tau_p(i+1, j+1)$ puts $k-1$ and k together and removes one p-breakpoint. If $-(k-1)$ exists, then either it is a singleton or it is located at the beginning of some strip. In both cases, a prefix reversal $\bar{\rho}_p(j-1)$ puts k and $-(k-1)$ together and removes one p-breakpoint. □

Lemma 21. *Let π be a signed permutation of the form $\pi = (\ldots\ldots\ \pi_{n-i}\ 1\ 2\ \ldots\ i)$, for $1 \leq i < n$ and $\pi_{n-i} \neq n$, where $s = \langle 1\ 2\ \ldots\ i \rangle$ is the last strip. During the execution of 2-P$\bar{\text{R}}$T, s will be removed from its location only when the element n is sent to position n.*

Proof. Since $\pi_1 \neq 1$, at each iteration 2-P$\bar{\text{R}}$T will try to remove two or one p-breakpoints. We describe now what happens in each one of these cases.

If a transposition $\tau_p(i, j)$ removes two p-breakpoints, then we must have $\pi_i = \pi_j - 1$. If s is involved in this transposition, then necessarily $j = n+1$. But then $\pi_{i-1} = n$, which is sent to position n.

Let π_i be the last element of the first strip of π and let $\tau_p(i, j)$ be a transposition that removes one p-breakpoint. So, we have $\pi_{j-1} = \pi_1 - 1$. If s is involved in this transposition, then π_{j-1} should be the last element of s, which would mean that $j = n+1$. However, in this case, s would only be increasing in length and it would not be removed from there.

Finally, if a reversal $\bar{\rho}_p(j)$ removes one p-breakpoint, we must have $\pi_{j+1} = -\pi_1 + 1$. If s is involved in this reversal, one must have $j = n$, in which case $\pi_1 = -n$ and n is sent to position n. □

Lemma 22. *Let π be a signed permutation of the form $\pi = (\ldots\ldots\ n\ 1\ 2\ \ldots\ i)$, for $1 \leq i < n$, where $s = \langle 1\ 2\ \ldots\ i \rangle$ is the last strip. During the execution of 2-P$\bar{\text{R}}$T, elements n and 1 remain adjacent until π can be sorted by one transposition.*

Proof. We have that $\pi_1 \neq 1$, so 2-P$\bar{\text{R}}$T will first try to apply a 2-move $\tau_p(i, j)$. As we mentioned, in this 2-move we must have $\pi_i \neq 1$. Therefore, if $\tau_p(i, j)$ separates n and 1, we should have that $\pi_j = 1$ and $\pi_{j-1} = n$. However, π_1 would have to be $n+1$, which is impossible.

Now we will try to apply a 1-move $\tau_p(i, j)$ by increasing the length of the first strip, which ends at π_{i-1}. To separate n and 1, we could have two possibilities. First, if $\pi_i = 1$, it would mean that the first strip ends at n and this transposition would sort π and remove two p-breakpoints at once. Second, if $\pi_j = 1$, then either π_{i-1} would have to be 0 or π_1 would have to be $n+1$, which is impossible.

Finally, we will try to perform a 1-move of the form $\bar{\rho}_p(j)$. To separate n and 1, we would need to have $\pi_{j+1} = 1$, which would mean that $\pi_1 = 0$, also impossible.

We can see that the only way that 2-PR̄T can separate elements 1 and n is if the permutation is of the form $((i+1)\ (i+2)\ \ldots\ n\ 1\ 2\ \ldots\ i)$. In this case, one transposition sorts it. □

Lemma 23. *During the execution of 2-PR̄T, π_1 will be 1 at most twice.*

Proof. When $\pi_1 = 1$ and $\pi_n \neq n$, the first strip is sent to the end of the permutation by 2-PR̄T. As Lemma 21 shows, it will only be removed from there when n goes to position n. Since 2-PR̄T is always trying to remove breakpoints, n will not be removed from the end until $\pi_1 = 1$ again. At this point, $\pi = (1\ 2\ \ldots\ i\ \pi_{i+1}\ \ldots\ldots\ n)$, so elements n and 1 are put together because the first strip is placed at position n. However, the algorithm will not separate them until the permutation is sorted, as shown by Lemma 22. □

Lemma 24. *For any signed permutation π, 2-PR̄T$(\pi, n) \leq b_p(\pi) + 2$.*

Proof. Starting from π, while the first element is not 1, 2-PR̄T always applies 2-moves or 1-moves. Besides, 1-moves are always possible, as Lemma 20 shows. At worse, it only applies 1-moves until some permutation π', for which $\pi'_1 = 1$, is reached. So, 2-PR̄T$(\pi, n) \leq (b_p(\pi) - b_p(\pi')) +$ 2-PR̄T(π', n).

If $\pi' = \iota_n$, the sorting ends and 2-PR̄T$(\pi, n) \leq b_p(\pi)$. Otherwise, since $\pi'_1 = 1$, we put the first strip in the end. There are two possibilities: $\pi'_n \neq n$ or $\pi'_n = n$.

If $\pi'_n \neq n$, moving the first strip to the end does not create nor remove a p-breakpoint. Suppose it generates π''. So, 2-PR̄T$(\pi, n) \leq (b_p(\pi) - b_p(\pi')) + 1 +$ 2-PR̄T(π'', n).

Now suppose that from π'' we keep applying 2-moves or 1-moves until π''', for which $\pi'''_1 = 1$, is reached. We have 2-PR̄T$(\pi, n) \leq (b_p(\pi) - b_p(\pi')) + 1 + (b_p(\pi'') - b_p(\pi''')) +$ 2-PR̄T(π''', n).

If $\pi''' = \iota_n$, the sorting ends and 2-PR̄T$(\pi, n) \leq (b_p(\pi) - b_p(\pi')) + 1 + b_p(\pi'') \leq b_p(\pi) + 1$, because $b_p(\pi') = b_p(\pi'')$. Otherwise, we must also have $\pi'''_n = n$, as Lemma 23 shows, and sending the first strip to the end creates one p-breakpoint and generates π''''. We have 2-PR̄T$(\pi, n) \leq (b_p(\pi) - b_p(\pi')) + 1 + (b_p(\pi'') - b_p(\pi''')) + 1 +$ 2-PR̄T(π'''', n).

Again, we will keep applying 2-moves or 1-moves on π'''' until a permutation of the form $(k\ k{+}1\ \ldots\ n\ 1\ 2\ \ldots\ k{-}1)$, which has two p-breakpoints, is reached, as Lemma 22 shows. Since the transposition to sort this permutation is a 2-move, 2-PR̄T$(\pi'''', n) \leq b_p(\pi'''') - 1$. We have 2-PR̄T$(\pi, n) \leq (b_p(\pi) - b_p(\pi')) + 1 + (b_p(\pi'') - b_p(\pi''')) + 1 + (b_p(\pi'''') - 1)$.

Since $b_p(\pi') = b_p(\pi'')$ and $b_p(\pi''') = b_p(\pi'''') - 1$, we have that 2-PR̄T$(\pi, n) \leq (b_p(\pi) - b_p(\pi')) + 1 + (b_p(\pi') - b_p(\pi''')) + 1 + (b_p(\pi''') + 1 - 1) = b_p(\pi) + 2$.

Now for the second possibility, suppose $\pi'_n = n$. So, moving the first strip of π' to the end creates one p-breakpoint. Suppose it generates π''. So, 2-PR̄T$(\pi, n) \leq (b_p(\pi) - b_p(\pi')) + 1 +$ 2-PR̄T(π'', n). Now, we will keep applying 2-moves or 1-moves on π'' until a permutation of the form $(k\ k+1\ \ldots\ n\ 1\ 2\ \ldots\ k-1)$ is reached, as Lemma 22 shows. Since the transposition to sort this permutation is a 2-move, 2-PR̄T$(\pi'', n) \leq b_p(\pi'') - 1$. Since $b_p(\pi') = b_p(\pi'') - 1$, we have that 2-PR̄T$(\pi, n) \leq (b_p(\pi) - b_p(\pi')) + 1 + (b_p(\pi'') - 1) = b_p(\pi) + 1$. □

Theorem 5. SbPR̄T *is $(2 + 4/b_p(\pi))$-approximable.*

Proof. From Lemmas 1 and 24, we have that the theoretical approximation factor for 2-PS$\bar{\text{R}}$T is $(b_p(\pi)+2)/(b_p(\pi)/2) = 2 + 4/b_p(\pi)$. □

Now we discuss the algorithm we propose for SbPS$\bar{\text{R}}$T, which we call 2-PS$\bar{\text{R}}$T. It works as follows while the permutation is not sorted: we try to apply a 2-move to π and, if this is not possible, we apply a 1-move, which is always possible, as shown in Lemma 25. Since there might exist more than one 1-move, we break ties by choosing the one that involves the least number of elements. If there is still a tie, we choose the prefix one.

Lemma 25. *Let π be any signed permutation with at least one psrt-breakpoint. For* SbPS$\bar{\text{R}}$T*, it is always possible to apply a* 1*-move on π.*

Proof. Consider the first element $\pi_1 = k$ of the permutation and let π_i be the last element of the first strip of π. We must have that either $\pi_j = dec(k,n)$ or $\pi_j = -dec(k,n)$ for some $j > 1$. If $dec(k,n)$ exists in π, then either it is a singleton, or it is at the end of some strip. In both cases, a transposition $\tau_p(i+1, j+1)$ puts $\pi_j = dec(k,n)$ and $\pi_1 = k$ together and removes one psrt-breakpoint. If $-dec(k,n)$ exists in π, then either it is a singleton, or it is at the beginning of some strip. In both cases, a prefix reversal $\bar{\rho}_p(j-1)$ puts $\pi_1 = k$ and $\pi_j = -dec(k,n)$ together and removes one psrt-breakpoint.

Consider the last element $\pi_n = k$ of the permutation and let π_j be the first element of the last strip of π. Now we must have that either $\pi_i = inc(k,n)$ or $\pi_i = -inc(k,n)$ for some $i < n$. If $inc(k,n)$ exists in π, then either it is a singleton, or it is at the beginning of some strip. In both cases, a transposition $\tau_s(i,j)$ puts $\pi_i = inc(k,n)$ and $\pi_1 = k$ together and removes one psrt-breakpoint. If $-inc(k,n)$ exists in π, then either it is a singleton, or it is at the end of some strip. In both cases, a suffix reversal $\rho_s(i+1)$ puts $\pi_1 = k$ and $\pi_i = -inc(k,n)$ together and removes one psrt-breakpoint. □

We point out that a 2-move for SbPS$\bar{\text{R}}$T can be found the same way that a 2-move for SbPST can be found, which can be seen in Lemma 8. However, one must be careful with the fact that $dec(\pi_1, n)$, for instance, not necessarily exists in π. We show 2-PS$\bar{\text{R}}$T in Algorithm 6.

Example 14. *The following example shows the execution of* 2-PS$\bar{\text{R}}$T *over* $\pi = (-9\ -7\ -8\ -15\ 4\ -12\ 6\ 3\ -14\ -11\ -5\ -13\ 10\ 2\ -1)$*:*

$\pi = (-9\ -7\ -8\ -15\ 4\ -12\ 6\ 3\ -14\ -11\ -5\ -13\ 10\ 2\ -1)$ // *no 2-move possible; $-dec(\pi_1, n)$ exists*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_p(12) = (13\ 5\ 11\ 14\ -3\ -6\ 12\ -4\ 15\ 8\ 7\ 9\ 10\ 2\ -1)$ // *$-inc(\pi_n, n)$ exists (least # of elements)*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_s(10) = (13\ 5\ 11\ 14\ -3\ -6\ 12\ -4\ 15\ 1\ -2\ -10\ -9\ -7\ -8)$ // *$inc(\pi_n, n)$ exists*
$\pi \leftarrow \pi{\cdot}\tau_s(14,15) = (13\ 5\ 11\ 14\ -3\ -6\ 12\ -4\ 15\ 1\ -2\ -10\ -9\ -8\ -7)$ // *2-move with suffix*
$\pi \leftarrow \pi{\cdot}\tau_s(6,11) = (13\ 5\ 11\ 14\ -3\ -2\ -10\ -9\ -8\ -7\ -6\ 12\ -4\ 15\ 1)$ // *$-inc(\pi_n, n)$ exists*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_s(7) = (13\ 5\ 11\ 14\ -3\ -2\ -1\ -15\ 4\ -12\ 6\ 7\ 8\ 9\ 10)$ // *2-move with suffix*
$\pi \leftarrow \pi{\cdot}\tau_s(3,11) = (13\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 14\ -3\ -2\ -1\ -15\ 4\ -12)$ // *$-inc(\pi_n, n)$ exists*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_s(9) = (13\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ -4\ 15\ 1\ 2\ 3\ -14)$ // *$dec(\pi_1, n)$ exists*
$\pi \leftarrow \pi{\cdot}\tau_p(2,10) = (5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ -4\ 15\ 1\ 2\ 3\ -14)$ // *$-inc(\pi_n, n)$ exists*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_s(10) = (5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ -3\ -2\ -1\ -15\ 4)$ // *$dec(\pi_1, n)$ exists*
$\pi \leftarrow \pi{\cdot}\tau_p(15,16) = (4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ -3\ -2\ -1\ -15)$ // *$-inc(\pi_n, n)$ exists*
$\pi \leftarrow \pi{\cdot}\bar{\rho}_s(12) = (4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 1\ 2\ 3)$ // *$\pi = \varphi^a_{k,n}$ for $k = 4$*
$\pi \leftarrow \pi{\cdot}\tau_p(13,16) = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$

Lemma 26 and Theorem 6 show that an approximation factor of $2 + 4/b_{psrt}(\pi)$ is guaranteed by 2-PS$\bar{\text{R}}$T.

Lemma 26. *For any signed permutation π, 2-PS$\bar{\text{R}}$T$(\pi, n) \leq b_{psrt}(\pi) + 2$.*

Algorithm 6 A $(2 + 4/b_{psrt}(\pi))$-approximation algorithm for SbPS$\bar{\text{R}}$T.

2-PS$\bar{\text{R}}$T(π, n)

Input: permutation π and its size n
Output: number of rearrangements used to sort π

$d \leftarrow 0$
while $\pi \neq \iota_n$ **do**
  if $\pi = \bar{\eta_n}$ **then**
    $\pi \leftarrow \pi \cdot \bar{\rho}_p(n)$
    $d \leftarrow d + 1$
  else if $\tau_p(i, j)$ is a 2-move with $\pi_{j-1} = dec(\pi_1, n)$ and $\pi_{i-1} = dec(\pi_j, n)$ **then**
    $\pi \leftarrow \pi \cdot \tau_p(i, j)$
    $d \leftarrow d + 1$
  else if $\tau_s(i, j)$ is a 2-move with $\pi_i = dec(\pi_n, n)$ and $\pi_j = dec(\pi_{i-1}, n)$ **then**
    $\pi \leftarrow \pi \cdot \tau_s(i, j)$
    $d \leftarrow d + 1$
  else // Otherwise, apply a 1-move
    // First, find all possible positions to apply a 1-move with prefix
    if $dec(\pi_1, n)$ exists in π **then**
      $j_1 \leftarrow \pi^{-1} dec(\pi_1, n) + 1$
      Let π_{i_1} be the last element of the first strip of π
    else
      $j_2 \leftarrow \pi^{-1} - dec(\pi_1, n) - 1$
    // Find all possible positions to apply a 1-move with suffix
    if $inc(\pi_n, n)$ exists in π **then**
      $i_3 \leftarrow \pi^{-1} inc(\pi_n, n)$
      Let π_{j_3} be the first element of the last strip of π
    else
      $i_4 \leftarrow \pi^{-1} - inc(\pi_n, n) + 1$
    Let λ be the rearrangement that involves the least number of elements between $\tau_p(i_1 + 1, j_1)$, $\bar{\rho}_p(j_2)$, $\tau_s(i_3, j_3)$, and $\bar{\rho}_s(i_4)$ (for i_x and j_y that are defined)
    if λ exists **then**
      $\pi \leftarrow \pi \cdot \lambda$
      $d \leftarrow d + 1$
    else if $\pi = \varphi^a_{k,n}$ for some $2 \leq k \leq n$ **then**
      $\pi \leftarrow \pi \cdot \tau_p(\pi_n^{-1} + 1, n + 1)$
      $d \leftarrow d + 1$
    else // $\pi = \varphi^s_{k,n}$ for some $2 \leq k \leq n$
      $\pi \leftarrow \pi \cdot \tau_p(\pi_n^{-1}, n + 1) \cdot \bar{\rho}_p(n)$
      $d \leftarrow d + 2$
return d

Proof. According to Lemma 25, it is always possible to apply a 1-move in π. When π has no psrt-breakpoints, by the definition of breakpoints it must be of one of the four forms: ι_n, $\bar{\eta}_n$, $\varphi^a_{k,n} = (k\ k{+}1\ k{+}2\ \ldots\ n\ 1\ 2\ 3\ \ldots\ k{-}1)$, or $\varphi^s_{k,n} = (-(k{-}1)\ -(k{-}2)\ -(k{-}3)\ \ldots\ -1\ -n\ -(n{-}1)\ -(n{-}2)\ \ldots\ -k)$, for $2 \leq k \leq n$ in both cases.

It is easy to note that ι_n is sorted ($d_{ps\bar{r}t}(\iota_n) = 0$), $\bar{\eta}_n$ is one reversal distant from being sorted ($d_{ps\bar{r}t}(\bar{\eta}_n) = 1$), $\varphi^a_{k,n}$ is one transposition distant from being sorted ($d_{ps\bar{r}t}(\varphi^a_{k,n}) = 1$), and $\varphi^s_{k,n}$ is one transposition and one reversal distant from being sorted ($d_{ps\bar{r}t}(\varphi^s_{k,n}) = 2$).

Therefore, in at most $b_{psrt}(\pi)$ iterations 2-PSR̄T ends up with a permutation with 0 psrt-breakpoints and at most two extra rearrangements will finish the sorting. □

Theorem 6. SbPSR̄T *is* $(2 + 4/b_{psrt}(\pi))$*-approximable.*

Proof. From Lemmas 1 and 26, we have that the theoretical approximation factor for 2-PSR̄T is $(b_{psrt}(\pi) + 2)/(b_{psrt}(\pi)/2) = 2 + 4/b_{psrt}(\pi)$. □

The algorithms 2-PR̄T and 2-PSR̄T presented in this section can also be easily implemented to run in $O(n^2)$ time or in $O(n \log n)$ time with the structure of log-lists [52].

4.6 Improving the Results in Practice

All the algorithms described in Sections 4.1 to 4.5 have a similar behavior: when we are trying to remove a certain amount of breakpoints, we search for the first opportunity to do so. For instance, consider 2-PSR̄, which was described in Section 4.4. Our first step in this algorithm is trying to remove one ps-breakpoint from π with one prefix reversal by placing π_1 before $-\pi_1 + 1$. If this is not possible, we try to remove one ps-breakpoint with a suffix reversal by placing π_n after $-\pi_n - 1$. If the two possibilities of removing one ps-breakpoint exist, we always use the prefix reversal. The same occurs with the several ways of removing one ps-breakpoint with two reversals. In fact, all other algorithms present, in a sense, this feature.

Now note that 2-PSR̄ sorts the permutation $\pi = (2\ 3\ {-4}\ 5\ 1)$ with 6 rearrangements: $\bar{\rho}_p(3) \cdot \bar{\rho}_p(1) \cdot \bar{\rho}_p(3) \cdot \bar{\rho}_p(5) \cdot \bar{\rho}_s(2) \cdot \bar{\rho}_p(1)$. The two first rearrangements were chosen because π has a $\pi_i = 3$ and a $\pi_j = -\pi_i - 1 = -4$ such that $i < j$ (item 3 of Lemma 15). However, π also has a $\pi_j = 5$ and a $\pi_i = -\pi_j + 1 = -4$ such that $i < j$ (item 4 of Lemma 15), which can be put together with two suffix reversals. Furthermore, when these suffix reversals are performed over π first, the sorting can be done with 5 rearrangements: $\bar{\rho}_s(3) \cdot \bar{\rho}_s(5) \cdot \bar{\rho}_p(2) \cdot \bar{\rho}_s(4) \cdot \bar{\rho}_p(3)$.

Considering this, we developed new algorithms, which in theory have the same approximation factors than the previous algorithms, but in practice will have the chance to find better results. They use the already existing algorithms to help them to decide which rearrangement to perform, as we explain below. They are algorithms for SbPR, SbPSR, SbPT, SbPST, SbPRT, SbPSRT, SbPSR̄, SbPR̄T, and SbPSR̄T and are called, respectively, 2-PRx, 2-PSRx, 2-PTx, 2-PSTx, 2-PRTx, 2-PSRTx, 2-PSR̄x, 2-PR̄Tx, and 2-PSR̄Tx.

The new algorithms act similarly and their idea can be seen as a general heuristic for genome rearrangement problems. Previous works have already shown some heuristics

for this kind of problems [21, 22], but in our case we guarantee the approximation factor, since we still use the greedy idea of removing the maximum amount of breakpoints at each iteration. In general, each one of the new algorithms searches for all the rearrangements that can remove the maximum amount of breakpoints from the current permutation π. Each one of these rearrangements is then performed over π and, among all the resulting permutations π', the adequate approximation algorithm is applied (for instance, `2-PSRTx` uses `2-PSRT`) so that we obtain an upper bound on the distance of π'. Note that there is no need for doing this if there is only one rearrangement that removes the maximum amount of breakpoints. After that, we choose the permutation π' whose upper bound on the distance is the smallest to be the next permutation of the sorting.

Example 15. *The following example shows the execution of* `2-PSRTx` *over* π = (9 6 2 14 10 15 7 12 4 11 3 8 13 1 5)*:*

π = (9 6 2 14 10 15 7 12 4 11 3 8 13 1 5) // *2-move with suffix (*`2-PSRT` *chooses prefix first)*
$\pi \leftarrow \pi{\cdot}\tau_s(2,12)$ = (9 8 13 1 5 6 2 14 10 15 7 12 4 11 3) // *2-move with prefix*
$\pi \leftarrow \pi{\cdot}\tau_p(9,10)$ = (10 9 8 13 1 5 6 2 14 15 7 12 4 11 3) // *2-move with prefix*
$\pi \leftarrow \pi{\cdot}\tau_p(14,15)$ = (11 10 9 8 13 1 5 6 2 14 15 7 12 4 3) // *2-move with suffix*
$\pi \leftarrow \pi{\cdot}\tau_s(9,12)$ = (11 10 9 8 13 1 5 6 7 12 4 3 2 14 15) // *2-move with suffix (n and* 1 *are an adjacency)*
$\pi \leftarrow \pi{\cdot}\tau_s(6,10)$ = (11 10 9 8 13 12 4 3 2 14 15 1 5 6 7) // *1-move with prefix transposition*
$\pi \leftarrow \pi{\cdot}\tau_p(5,7)$ = (13 12 11 10 9 8 4 3 2 14 15 1 5 6 7) // *1-move with prefix reversal*
$\pi \leftarrow \pi{\cdot}\rho_p(9)$ = (2 3 4 8 9 10 11 12 13 14 15 1 5 6 7) // *2-move with prefix*
$\pi \leftarrow \pi{\cdot}\tau_p(4,13)$ = (8 9 10 11 12 13 14 15 1 2 3 4 5 6 7) // $\pi = \varphi^a_{k,n}$ *for* $k = 8$
$\pi \leftarrow \pi{\cdot}\tau_p(9,16)$ = (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)

In Example 11, we showed `2-PSRT` *executing over the same permutation. It used 10 rearrangements to sort it there.*

All these new algorithms have time complexity $O(n^4)$ or $O(n^3 \log n)$: the main loop is executed while the permutation is not sorted, and we know that the distance is $O(n)$; at each loop, we can have $O(n)$ possible rearrangements that remove the maximum amount of breakpoints; for each of these rearrangements, we apply the approximation algorithms showed in the previous sections, which have $O(n^2)$ or $O(n \log n)$ time.

4.7 Experimental Results

All the algorithms presented in Sections 4.1 to 4.6 as well as algorithms `2-PR`, `2-PT`, `2-PRT`, and `2-PR`$\bar{\text{R}}$ were implemented in C language and executed over the same instances. We created *Set U1*, which contains all 4,037,912 unsigned permutations of size n for $2 \leq n \leq 10$, *Set U2*, which contains 1,990,000 unsigned permutations, *Set S1*, which contains all 196,811,960 signed permutations of size n for $2 \leq n \leq 9$, and *Set S2*, which also contains 1,990,000 signed permutations. Sets U2 and S2 contain 10,000 different and randomly generated permutations of each size n, with n varying between 10 and 1,000 in intervals of 5; all these permutations only contain singletons (considering the most restrict kind of breakpoints, i. e., upsr-breakpoints), which means that they always have the maximum amount of breakpoints and should be more difficult to be sorted.

As we mentioned at the end of Section 4.6, the algorithms that we presented in that section have time complexity $O(n^4)$. Because of this, considering Sets U2 and S2, the algorithms were executed over all the 590,000 permutations of sizes up to 300, over 40,000

Table 4.1: Average approximation factors (approx.) and average amount of rearrangements (# rear.) used by algorithms 2-PSR, 2-PST, and 2-PSRT when the number of permutations tested for each n increases as n increases.

Alg.	n	Number of permutations					
		$10,000$		$1,000 \times n$		$10,000 \times n$	
		approx.	# rear.	approx.	# rear.	approx.	# rear.
2-PSR	10	1.149	10.339	1.149	10.339	1.149	10.341
	50	1.133	55.536	1.133	55.516	1.133	55.511
	100	1.124	111.298	1.124	111.311	1.125	111.327
	150	1.122	167.126	1.121	167.045	1.121	167.033
	200	1.119	222.607	1.119	222.667	1.119	222.688
	250	1.118	278.314	1.118	278.295	1.118	278.326
2-PST	10	1.327	5.971	1.327	5.971	1.328	5.977
	50	1.317	32.267	1.317	32.255	1.316	32.249
	100	1.311	64.905	1.311	64.884	1.311	64.890
	150	1.307	97.339	1.307	97.360	1.307	97.363
	200	1.305	129.798	1.304	129.738	1.304	129.732
	250	1.302	162.070	1.301	162.044	1.302	162.040
2-PSRT	10	1.553	6.813	1.553	6.813	1.554	6.817
	50	1.372	33.595	1.373	33.605	1.373	33.598
	100	1.353	66.939	1.353	66.935	1.353	66.944
	150	1.347	100.309	1.346	100.282	1.346	100.278
	200	1.343	133.621	1.343	133.599	1.343	133.608
	250	1.341	166.913	1.341	166.931	1.341	166.926

permutations of sizes between 305 and 500 (1,000 for each size n), and over 10,000 permutations of sizes between 505 and 1,000 (100 for each size n). For Sets U1 and S1, we were able to execute the algorithms over all permutations. In order to perform a fair comparison, however, for the algorithms in Section 4.6 we will consider only the results regarding permutations of size up to 300.

The experimental results for all these algorithms are shown in Figures 4.3 to 4.7. The x-axis represents the values of n while the y-axis represents the average of the approximation factor between the permutations of that size. For Sets U1 and S1, the approximation factors were calculated using the exact distances of the permutations and for Sets U2 and S2 they were calculated using the theoretical lower bounds of the distances, which were given in Lemmas 1 and 2.

Note that 10,000 permutations for each size of n in Sets U2 and S2 is a small number when we think of $n!$ of $2^n n!$. However, we point out in Table 4.1 that larger amounts of permutations for each n do not change considerably the average approximation factors nor the average amount of rearrangements used to sort the permutations. For this reason, we decided to keep all tests with $10,000$ random permutations for each size n.

Since some of the algorithms that we developed are $(2 + B)$-approximations, where B is a constant divided by the number of breakpoints, it was expected that sometimes their approximation factor were above 2 (but never surpassing, of course, the theoretical upper bounds we proved). This happened for 2-PSRT in 0.027% of the permutations of Set U1 (for all $4 \leq n \leq 10$) and in 0.001% of the permutations of Set U2 (for $n = 10$ only), for 2-PS$\bar{\text{R}}$ in one permutation of Set S2 (for $n = 10$), for 2-P$\bar{\text{R}}$T in 0.314% of the permutations of Set S2 (for $10 \leq n \leq 85$), for 2-PS$\bar{\text{R}}$T in 0.001% of the permutations of Set S1 (for $2 \leq n \leq 9$) and in 0.445% of the permutations of Set S2 (for $10 \leq n \leq 75$), for 2-PRTx in 0.001% of the permutations of Set U1 (for $5 \leq n \leq 10$), for 2-P$\bar{\text{R}}$Tx in 0.006% of the permutations of Set S2 (for $n = 10$ and $n = 15$ only), and for 2-PS$\bar{\text{R}}$Tx in 0.003%

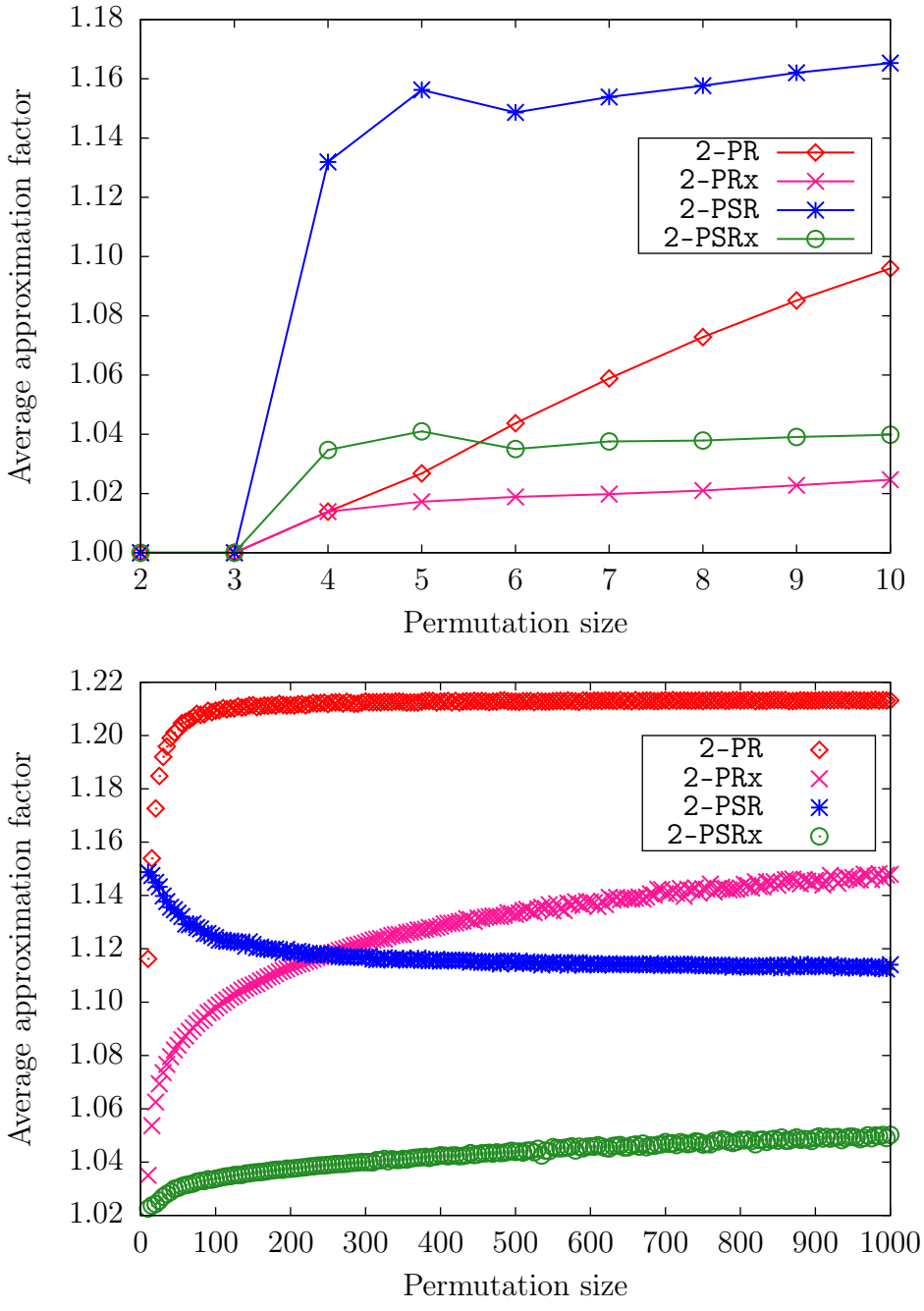


Figure 4.3: Average approximation factors for `2-PR`, `2-PRx`, `2-PSR`, and `2-PSRx` when the permutation size grows.

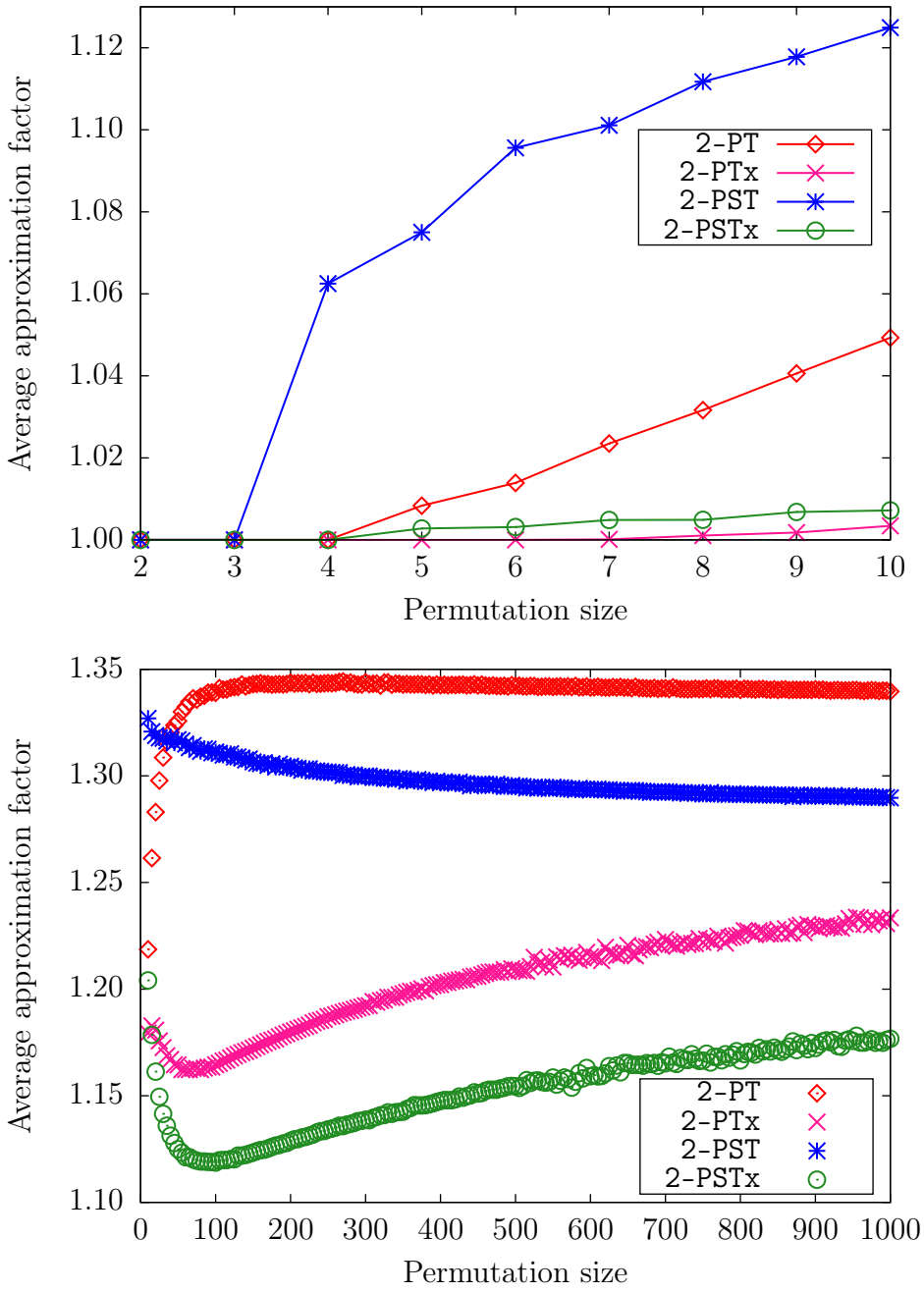


Figure 4.4: Average approximation factors for `2-PT`, `2-PTx`, `2-PST`, and `2-PSTx` when the permutation size grows.

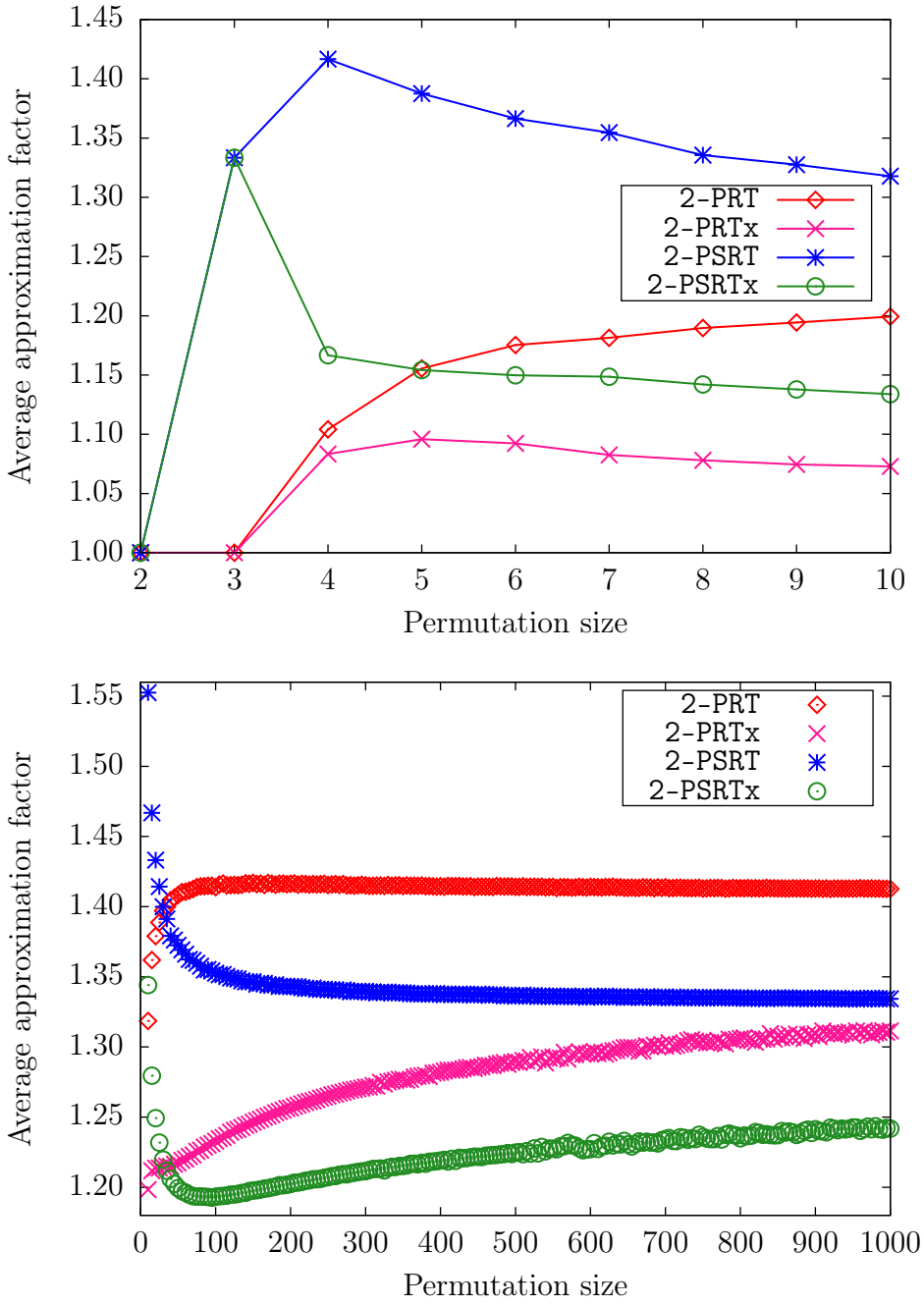


Figure 4.5: Average approximation factors for `2-PRT`, `2-PRTx`, `2-PSRT`, and `2-PSRTx` when the permutation size grows.

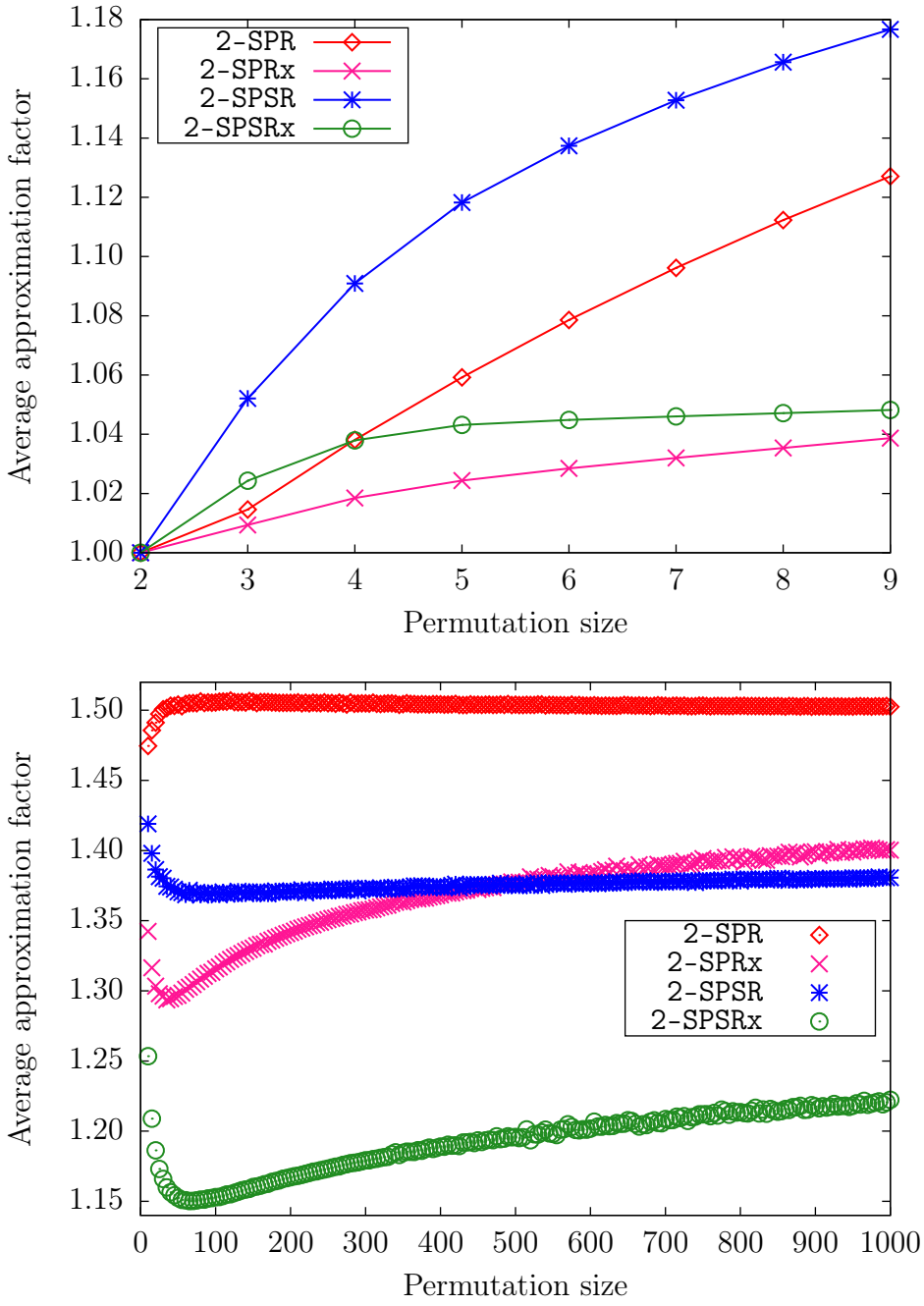


Figure 4.6: Average approximation factors for 2-P$\bar{\text{R}}$, 2-P$\bar{\text{R}}$x, 2-PS$\bar{\text{R}}$, and 2-PS$\bar{\text{R}}$x when the permutation size grows.

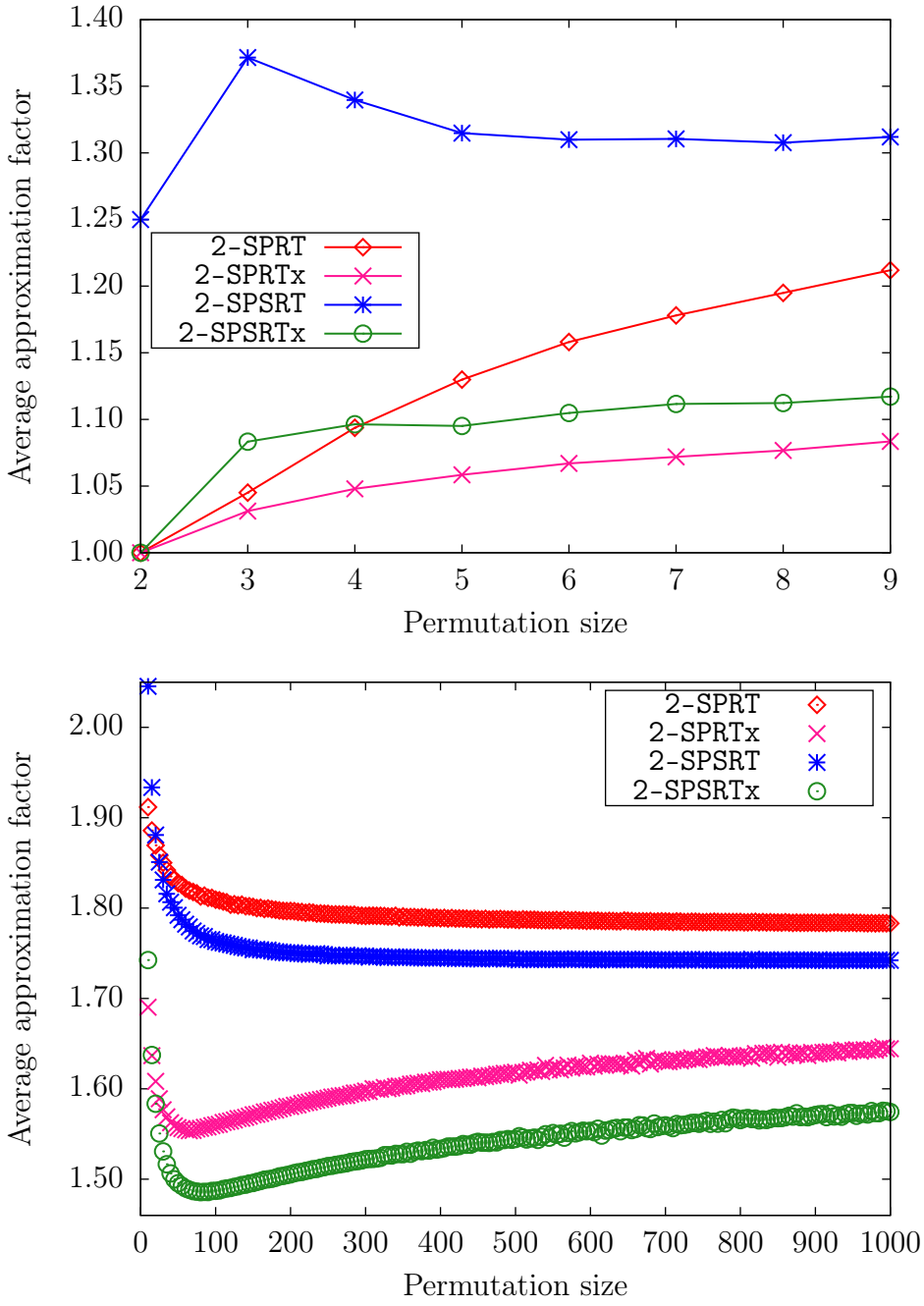


Figure 4.7: Average approximation factors for 2-PR̄T, 2-PR̄Tx, 2-PSR̄T, 2-PSR̄Tx when the permutation size grows.

of the permutations of Set S2 (for $n = 10$ and $n = 15$ only).

Table 4.2 shows the maximum approximation factors computed for each value of n and each algorithm. Regarding the factors obtained for Sets U2 and S2, we give them only for some values of n. One must remember that these last ones were computed using the theoretical lower bound on the distances of the problems and, although they do not show the real approximation factors and cannot really demonstrate how close to the theoretical factors the algorithms might be, they can help us compare the algorithms. In this table, we can see that the maximum factor obtained by `2-PSRT` over Set U1 is 3. This happened, for instance, with permutation $\pi = (1\ 2\ 5\ 4\ 3)$. Clearly, $d_{psrt}(\pi) = 1$, but `2-PSRT` uses $\rho_p(2)$, $\tau_p(3, 6)$, and $\rho_p(5)$ to sort it. Despite the fact that both $\rho_p(2)$ and $\rho_s(3)$ can remove one upsr-breakpoint each, our implementation decision was to choose the one that involves less elements.

We expected that algorithms that allow only prefix rearrangements would usually return bigger values for the size of the sorting sequences when compared to the algorithms that allow suffix rearrangements along with prefix rearrangements, because that is what happens with the real distances, as we showed in Chapter 1. In fact, by looking at the graphics of Figures 4.3 to 4.7 we can see that this indeed happens for all the permutations of Sets U2 and S2. However, we can also see that it does not happen for the permutations of Sets U1 and S1. In order to establish a more fair comparison, we individually checked the number of rearrangements returned by the algorithms for each permutation. We say that an algorithm X is *better* than another algorithm Y for a permutation π if $X(\pi) < Y(\pi)$. In our comparisons we saw that, for example, considering $n = 10$ on Set U1, `2-PSR` was better than `2-PR` in 51.952% of the 10! permutations while `2-PR` was better than `2-PSR` in only 7.772% of these permutations. For this reason, we say that `2-PSR` was better than `2-PR` *more times than the contrary*.

What we found out in these individual comparisons was that all algorithms with both prefix and suffix rearrangements were better than the algorithms with only prefix rearrangements more times than the contrary, when considering all the permutations tested. We also point out that when $n = 1000$, `2-PSR`, `2-PST`, `2-PSRT`, `2-PSR̄`, and `2-PSR̄T` performed, in average, 100.32, 25.72, 39.84, 123.29, 21.66 less rearrangements than `2-PR`, `2-PT`, `2-PRT`, `2-PR̄`, and `2-PR̄T`, respectively.

The most probable cause for the behavior of the average approximation factors showed in the graphics for the permutations of Sets U1 and S1 is very simple: for small values of n, the algorithms tend to return similar results but the distances tend to decrease. For example, consider permutation $\pi = (6\ 3\ 4\ 5\ 1\ 2)$. We can check that $d_{pr}(\pi) = 5$ and $d_{psr}(\pi) = 3$. However, both `2-PR` and `2-PSR` sort π with 5 rearrangements. For this reason, the approximation factor for the former is smaller than the approximation factor for the latter.

When we consider only the average approximation factor of the algorithms for each value of n (as shown in the graphics of Figures 4.3 to 4.7), we can see that, for $n \geq 100$, the average factors for `2-PSR` are always below 1.124, for `2-PST` are always below 1.311, for `2-PSRT` are always below 1.353, for `2-PSR̄` are always below 1.381, for `2-PR̄T` are always below 1.810, and for `2-PSR̄T` are always below 1.764. For the algorithms presented in Section 4.6, the average factors are even smaller: for `2-PSRx` they are always below 1.041,

Table 4.2: Maximum approximation factors reached on all permutations tested of the same size n for each algorithm.

Algorithm	Sets U1 and S1, values for n								
	2	3	4	5	6	7	8	9	10
2-PR	1.000	1.000	1.333	1.333	1.400	1.600	1.714	1.857	1.857
2-PRx	1.000	1.000	1.333	1.333	1.400	1.500	1.500	1.571	1.571
2-PSR	1.000	1.000	1.500	2.000	2.000	2.000	2.000	2.000	2.000
2-PSRx	1.000	1.000	1.500	1.666	1.666	1.750	1.750	1.750	1.833
2-PT	1.000	1.000	1.000	1.333	1.333	1.500	1.500	1.600	1.600
2-PTx	1.000	1.000	1.000	1.000	1.000	1.200	1.200	1.200	1.333
2-PST	1.000	1.000	1.500	1.500	1.666	1.666	1.750	1.750	1.800
2-PSTx	1.000	1.000	1.000	1.333	1.333	1.333	1.333	1.500	1.500
2-PRT	1.000	1.000	1.500	2.500	2.500	2.500	2.500	2.500	2.500
2-PRTx	1.000	1.000	1.500	2.500	2.500	2.500	2.500	2.500	2.500
2-PSRT	1.000	2.000	3.000	3.000	3.000	3.000	3.000	3.000	3.000
2-PSRTx	1.000	2.000	2.000	2.000	2.000	2.000	2.000	2.000	2.000
2-P$\bar{\text{R}}$	1.000	1.250	1.400	1.666	1.714	1.750	1.777	1.800	
2-P$\bar{\text{R}}$x	1.000	1.250	1.333	1.500	1.500	1.555	1.555	1.600	
2-PS$\bar{\text{R}}$	1.000	1.333	1.750	1.800	1.833	1.857	1.875	1.888	
2-PS$\bar{\text{R}}$x	1.000	1.333	1.750	1.750	1.833	1.857	1.875	1.888	
2-P$\bar{\text{R}}$T	1.000	1.500	1.666	1.750	2.000	2.000	2.000	2.000	
2-P$\bar{\text{R}}$Tx	1.000	1.500	1.500	1.500	1.750	1.750	1.800	1.833	
2-PS$\bar{\text{R}}$T	3.000	3.000	3.000	3.000	3.000	3.000	3.000	3.000	
2-PS$\bar{\text{R}}$Tx	1.000	1.500	1.500	1.666	1.666	1.750	1.750	1.800	

Algorithm	Sets U2 and S2, values for n								
	25	50	75	100	125	250	500	750	1000
2-PR	1.480	1.420	1.413	1.350	1.328	1.304	1.276	1.265	1.264
2-PRx	1.200	1.180	1.160	1.160	1.152	1.152	1.160	1.158	1.168
2-PSR	1.500	1.346	1.364	1.323	1.298	1.269	1.264	1.236	1.235
2-PSRx	1.125	1.081	1.081	1.080	1.072	1.064	1.060	1.057	1.062
2-PT	1.680	1.600	1.600	1.560	1.536	1.496	1.444	1.426	1.414
2-PTx	1.360	1.320	1.280	1.260	1.248	1.248	1.256	1.248	1.258
2-PST	1.833	1.673	1.621	1.555	1.516	1.461	1.406	1.385	1.377
2-PSTx	1.333	1.265	1.216	1.232	1.209	1.196	1.198	1.198	1.213
2-PRT	1.840	1.680	1.626	1.600	1.584	1.536	1.512	1.482	1.470
2-PRTx	1.440	1.360	1.333	1.320	1.312	1.328	1.332	1.336	1.338
2-PSRT	1.833	1.632	1.567	1.555	1.516	1.461	1.422	1.409	1.391
2-PSRTx	1.416	1.306	1.297	1.313	1.274	1.269	1.266	1.260	1.271
2-P$\bar{\text{R}}$	1.840	1.760	1.733	1.690	1.680	1.628	1.590	1.574	1.561
2-P$\bar{\text{R}}$x	1.520	1.440	1.400	1.410	1.408	1.424	1.430	1.424	1.428
2-PS$\bar{\text{R}}$	1.833	1.693	1.648	1.646	1.620	1.574	1.555	1.544	1.539
2-PS$\bar{\text{R}}$x	1.375	1.285	1.256	1.232	1.233	1.240	1.228	1.241	1.247
2-P$\bar{\text{R}}$T	2.160	2.080	2.026	2.000	1.968	1.904	1.900	1.853	1.852
2-P$\bar{\text{R}}$Tx	1.920	1.720	1.706	1.700	1.680	1.680	1.676	1.674	1.682
2-PS$\bar{\text{R}}$T	2.173	2.081	2.054	1.979	1.967	1.887	1.843	1.839	1.825
2-PS$\bar{\text{R}}$Tx	1.833	1.673	1.648	1.636	1.612	1.630	1.607	1.602	1.615

for 2-PSTx they are always below 1.139, for 2-PSRTx they are always below 1.212, for 2-PS$\bar{\text{R}}$x they are always below 1.180, for 2-P$\bar{\text{R}}$Tx they are always below 1.598, and for 2-PS$\bar{\text{R}}$Tx they are always below 1.523 (recall we are considering n up to 300).

We also point out that the algorithms developed in Section 4.6 indeed used a number of rearrangements smaller than or equal to the number of rearrangements for each permutation when compared to the number of rearrangements returned by the previous algorithms, except for 2-PSRT vs. 2-PSRTx. Specifically, this happened only in Set U2 and for two permutations when $n \in \{20, 35\}$ and for one permutation when $n \in \{30, 45, 65, 85, 100, 140, 195\}$. Tables 4.3 and 4.4 show the average amount of rearrangements that the "x" algorithms used less than the other algorithms, for each size n. We reinforce that the x algorithms did not always used a strictly smaller amount of rearrangements when compared to the previous algorithms (they also used equal amounts), but when they did, they did it for a very large amount of permutations. For $n \geq 30$, 2-PRx, 2-PSRx, 2-PTx, 2-PSTx, 2-PRTx, 2-PSRTx, 2-P$\bar{\text{R}}$x, 2-PS$\bar{\text{R}}$x, 2-P$\bar{\text{R}}$Tx, and 2-PS$\bar{\text{R}}$Tx were better than 2-PR, 2-PSR, 2-PT, 2-PST, 2-PRT, 2-PSRT, 2-P$\bar{\text{R}}$, 2-PS$\bar{\text{R}}$, 2-P$\bar{\text{R}}$T, and 2-PS$\bar{\text{R}}$T, respectively, in more than 92.91%, 93.43%, 87.87%, 91.95%, 91.15%, 92.84%, 98.26%, 98.23%, 98.28%, and 98.79% of the permutations in each size n.

4.8 Bounds on the Diameters

In this section we present bounds for the diameters concerning the problems SbPSR, SbPST, SbPSRT, SbP$\bar{\text{R}}$T, SbPS$\bar{\text{R}}$, and SbPS$\bar{\text{R}}$T. We first show families of permutations which will be useful for that purpose.

For SbPSR, let

$$\pi_n^{psr} = \begin{cases} (n\ 1\ n{-}2\ n{-}4\ \ldots\ 4\ 2\ n{-}3\ n{-}5\ \ldots\ 3\ n{-}1) & \text{if } n \text{ is even} \\ (n\ 1\ n{-}2\ n{-}4\ \ldots\ 5\ 3\ n{-}3\ n{-}5\ \ldots\ 2\ n{-}1) & \text{if } n \text{ is odd.} \end{cases} \tag{4.1}$$

Lemma 27. *For $n \geq 8$, $n - 1 \leq d_{psr}(\pi_n^{psr}) \leq n$.*

Proof. The lower bound is valid because $d_{psr}(\pi) \geq b_{upsr}(\pi)$ for any π, according to Lemma 1, and $b_{upsr}(\pi_n^{psr}) = n - 1$.

Now we show that Algorithm 7 sorts π_n^{psr} with n rearrangements. The proofs for n odd and n even are similar, so we show here only the proof for n even. The loop invariant for the while loop in line 2 is the following: before the ith iteration, elements from $n{-}3{-}i$ to n are in the correct positions, and we are in one of the two following situations: (i) either i is odd and all other odd elements are in decreasing order in the beginning of the permutation followed by all other even elements in increasing order, or (ii) i is even and all other even elements are in decreasing order in the beginning of the permutation followed by all other odd elements in increasing order.

It is easy to note that before the first iteration $\pi = (n{-}5\ n{-}7\ \ldots\ 1\ 2\ 4\ 6\ \ldots\ n{-}4\ n{-}3\ n{-}2\ n{-}1\ n)$, because the six first reversals lead π_n^{psr} to it. Here, $i = 1$ and elements from $n{-}4$ to n are in the correct positions. Also, all other odd elements are in decreasing order in the beginning followed by all other even elements in increasing order.

Table 4.3: Average number of rearrangements that 2-PRx, 2-PSRx, 2-PTx, 2-PSTx, 2-PRTx, and 2-PSRTx performed less than 2-PR, 2-PSR, 2-PT, 2-PST, 2-PRT, and 2-PSRT, respectively (Sets U1 and U2).

n	2-PRx × 2-PR	2-PSRx × 2-PSR	2-PTx × 2-PT	2-PSTx × 2-PST	2-PRTx × 2-PRT	2-PSRTx × 2-PSRT
2	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.208	0.000	0.125	0.041	0.291
5	0.041	0.333	0.025	0.166	0.158	0.400
6	0.127	0.423	0.047	0.262	0.248	0.497
7	0.233	0.537	0.091	0.324	0.345	0.587
8	0.356	0.662	0.136	0.418	0.445	0.663
9	0.487	0.793	0.194	0.496	0.542	0.753
10	0.625	0.927	0.255	0.591	0.641	0.831
50	5.920	5.049	4.031	4.705	4.774	4.234
55	6.527	5.461	4.615	5.216	5.257	4.638
60	7.076	5.773	5.102	5.715	5.714	5.011
65	7.599	6.233	5.605	6.146	6.161	5.365
70	8.153	6.726	6.073	6.695	6.634	5.799
75	8.711	7.057	6.487	7.138	7.060	6.153
80	9.171	7.466	6.997	7.596	7.518	6.480
85	9.677	7.794	7.422	8.130	7.899	6.783
90	10.191	8.209	7.903	8.610	8.315	7.214
95	10.655	8.601	8.345	9.019	8.723	7.593
100	11.140	8.953	8.720	9.529	9.055	7.887
105	11.612	9.269	9.236	9.923	9.508	8.206
110	12.140	9.680	9.579	10.349	9.953	8.596
115	12.534	10.053	9.999	10.839	10.247	8.904
120	12.992	10.489	10.414	11.313	10.566	9.180
125	13.433	10.844	10.810	11.694	10.930	9.543
130	13.972	11.251	11.202	12.060	11.318	9.882
135	14.309	11.668	11.655	12.493	11.684	10.125
140	14.768	11.853	11.986	12.872	12.074	10.450
145	15.292	12.501	12.416	13.332	12.419	10.739
150	15.746	12.761	12.803	13.651	12.788	11.090
155	16.063	12.973	13.180	14.016	13.066	11.307
160	16.471	13.418	13.560	14.414	13.431	11.685
165	16.956	13.719	13.898	14.884	13.671	11.987
170	17.285	14.076	14.244	15.183	14.122	12.242
175	17.767	14.401	14.590	15.628	14.403	12.531
180	18.101	14.832	14.971	15.892	14.625	12.881
185	18.587	15.140	15.299	16.433	15.013	13.031
190	18.915	15.398	15.624	16.739	15.346	13.404
195	19.256	15.958	15.986	17.099	15.587	13.714
200	19.649	16.145	16.384	17.485	15.990	13.995
205	20.009	16.523	16.685	17.769	16.287	14.268
210	20.468	16.992	17.046	18.164	16.508	14.576
215	20.904	17.282	17.360	18.434	16.856	14.806
220	21.136	17.480	17.715	18.892	17.125	15.010
225	21.631	17.911	17.991	19.274	17.391	15.309
230	22.099	18.147	18.344	19.521	17.765	15.602
235	22.389	18.493	18.585	19.839	18.009	15.848
240	22.839	19.030	18.951	20.174	18.289	16.130
245	23.255	19.229	19.292	20.539	18.630	16.382
250	23.411	19.627	19.553	20.857	18.808	16.624
255	23.942	20.038	19.891	21.210	19.102	16.883
260	24.342	20.296	20.222	21.557	19.467	17.167
265	24.665	20.529	20.600	21.862	19.651	17.305
270	24.888	20.919	20.950	22.122	20.023	17.663
275	25.419	21.207	21.140	22.497	20.161	17.944
280	25.640	21.517	21.531	22.815	20.442	18.053
285	25.821	21.945	21.756	23.162	20.712	18.320
290	26.267	22.294	21.999	23.332	21.028	18.590
295	26.701	22.569	22.307	23.757	21.377	18.865
300	27.173	22.974	22.738	24.101	21.571	19.167

Table 4.4: Average number of rearrangements that 2-P$\bar{\text{R}}$x, 2-PS$\bar{\text{R}}$x, 2-P$\bar{\text{R}}$Tx, and 2-PS$\bar{\text{R}}$Tx performed less than 2-P$\bar{\text{R}}$, 2-PS$\bar{\text{R}}$, 2-P$\bar{\text{R}}$T, and 2-PS$\bar{\text{R}}$T, respectively (Sets S1 and S2).

n	2-P$\bar{\text{R}}$x × 2-P$\bar{\text{R}}$	2-PS$\bar{\text{R}}$x × 2-PS$\bar{\text{R}}$	2-P$\bar{\text{R}}$Tx × 2-P$\bar{\text{R}}$T	2-PS$\bar{\text{R}}$Tx × 2-PS$\bar{\text{R}}$T
2	0.000	0.000	0.000	0.250
3	0.020	0.083	0.041	0.479
4	0.101	0.205	0.153	0.585
5	0.218	0.368	0.283	0.701
6	0.373	0.554	0.423	0.803
7	0.555	0.755	0.567	0.918
8	0.758	0.966	0.714	1.037
9	0.977	1.186	0.863	1.163
50	10.364	10.688	6.764	7.264
55	11.241	11.826	7.412	7.949
60	12.203	12.851	8.006	8.624
65	13.153	14.178	8.639	9.303
70	14.044	15.137	9.240	9.931
75	14.886	16.244	9.772	10.573
80	15.834	17.146	10.270	11.202
85	16.556	18.346	10.899	11.864
90	17.391	19.308	11.416	12.431
95	18.250	20.400	11.975	13.070
100	18.986	21.465	12.453	13.662
105	19.856	22.406	13.005	14.280
110	20.572	23.612	13.443	14.853
115	21.319	24.418	13.965	15.430
120	22.154	25.545	14.367	16.020
125	22.739	26.451	14.914	16.568
130	23.460	27.616	15.339	17.117
135	24.199	28.468	15.952	17.636
140	24.824	29.304	16.358	18.207
145	25.702	30.455	16.844	18.760
150	26.327	31.227	17.245	19.231
155	26.798	32.346	17.689	19.820
160	27.638	33.295	18.219	20.353
165	28.446	34.002	18.534	20.816
170	28.991	34.927	19.106	21.436
175	29.636	35.947	19.464	21.929
180	30.229	37.023	19.935	22.390
185	30.863	37.767	20.314	22.858
190	31.483	38.633	20.707	23.380
195	32.102	39.664	21.062	23.865
200	32.744	40.572	21.636	24.401
205	33.344	41.422	22.020	24.858
210	33.967	42.386	22.436	25.310
215	34.439	43.484	22.777	25.838
220	35.152	43.872	23.124	26.293
225	35.627	45.085	23.649	26.776
230	36.438	45.814	23.966	27.278
235	36.817	46.840	24.268	27.867
240	37.492	47.721	24.716	28.271
245	38.155	48.476	25.111	28.704
250	38.551	49.329	25.484	28.954
255	39.145	50.439	25.793	29.586
260	39.708	51.194	26.250	29.986
265	40.363	51.930	26.618	30.466
270	40.872	52.692	27.043	30.943
275	41.125	53.607	27.510	31.549
280	42.201	54.647	27.777	31.805
285	42.423	55.536	28.093	32.271
290	43.103	55.927	28.469	32.840
295	43.587	57.026	28.840	33.215
300	44.194	57.992	29.079	33.618

Let i be even. Note that elements from $n-3-i$ to n are in the correct order, the first element is $n-4-i$, an even element, followed by all other even elements in decreasing order, followed by all other odd elements in increasing order. During the ith iteration, $\rho_p(\pi^{-1}_{\pi_1+1} - 1)$ is applied, which puts $n-4-i$ before $n-3-i$. This puts elements from $n-4-i$ to n in the correct position, puts all other odd elements in decreasing order in the beginning of the permutation followed by all other even elements in increasing order, which keeps the invariant valid for iteration $i+1$. If i is odd, the argument is similar.

Note that, for each iteration, element π_1 is one unit less than the element that was in π_1 in the previous iteration. Therefore, eventually we have $\pi_1 = 1$. Since for the first iteration we have $\pi_1 = n–5$, the while loop has exactly $n–6$ iterations. Before the $(n–6)$th iteration, which is an even number, we must have elements from $n-3-(n-6) = 3$ to n in the correct position and element 2 in the first position followed by element 1. Since $\rho_p(\pi^{-1}_{\pi_i+1} - 1)$ is applied, after this iteration the permutation will be sorted.

Since there are $n-6$ rearrangements on the while loop and 6 rearrangements previous to it, the algorithm uses a total of n rearrangements to sort π^{psr}_n, which means that $d_{psr}(\pi^{psr}_n) \leq n$. □

Algorithm 7 An algorithm to sort π^{psr}_n with prefix reversals and suffix reversals.

SORT_FAMILY_SBPSR(π, n)

Input: permutation $\pi = \pi^{psr}_n$ and its size $n \geq 8$
Output: number of rearrangements used to sort π

1 $\pi \leftarrow \pi \cdot \rho_p(n-1) \cdot \rho_p(n-3) \cdot \rho_s(2) \cdot \rho_s(\pi^{-1}_{\pi_n+1} + 1) \cdot \rho_p(\pi^{-1}_{\pi_1-1} - 1) \cdot \rho_s(\pi^{-1}_n)$
2 $d \leftarrow 6$
3 **while** $\pi_1 \neq 1$ **do**
4 $\quad \pi \leftarrow \pi \cdot \rho_p(\pi^{-1}_{\pi_1+1} - 1)$
5 $\quad d \leftarrow d + 1$
6 **return** d

For SBPST, let

$$\pi^{pst}_n = \eta_n = (n\ n-1\ n-2\ \ldots\ 2\ 1). \tag{4.2}$$

Lemma 28. *For $n \geq 3$, $\left\lceil \frac{n-1}{2} \right\rceil + 1 \leq d_{pst}(\pi^{pst}_n) \leq n - \left\lfloor \frac{n}{4} \right\rfloor$.*

Proof. First note that the lower bound is valid because $d_{pst}(\pi) \geq \lceil b_{ps}(\pi)/2 \rceil$ for any π, according to Lemma 1, and $b_{ps}(\pi^{pst}_n) = n - 1$. However, this lower bound is not tight, as we show next, and the distance of π^{pst}_n is at least $\lceil (n-1)/2 \rceil + 1$.

If $\lceil b_{ps}(\pi)/2 \rceil$ was a tight lower bound, then when n is even the sorting should have $\lceil (n-1)/2 \rceil - 1$ 2-moves and one 1-move. We can see that applying a 2-move in π^{pst}_n is already not possible. Also, the 1-moves that are possible will either place n after $n-1$ or place 1 before 2. In any case, it is not possible to apply a 2-move in the generated permutation. When n is odd, the sorting should only contain 2-moves, which is not possible.

The upper bound is valid because of the algorithm presented by Dias and Meidanis [24] that sorts η_n using at most $n - \lfloor n/4 \rfloor$ prefix transpositions. □

For SbPRT, let

$$\pi_n^{prt} = \begin{cases} (n{-}1\ n{-}2\ n\ n{-}4\ n{-}6\ \ldots\ 2\ n{-}3\ n{-}5\ \ldots\ 1) & \text{if } n \text{ is even} \\ (n\ n{-}3\ n{-}1\ n{-}5\ n{-}7\ \ldots\ 2\ n{-}2\ n{-}4\ \ldots\ 1) & \text{if } n \text{ is odd.} \end{cases} \tag{4.3}$$

Lemma 29. *For $n \geq 7$, $\left\lceil \frac{n}{2} \right\rceil \leq d_{prt}(\pi_n^{prt}) \leq \left\lceil \frac{n}{2} \right\rceil + 1$.*

Proof. First note that $d_{prt}(\pi_n^{prt}) \geq \lceil n/2 \rceil$ [39] because $b_{upr}(\pi_n^{prt}) = n - 1$ when n is even and $\lceil (n-1)/2 \rceil = \lceil n/2 \rceil$, and $b_{upr}(\pi_n^{prt}) = n$ when n is odd.

Now we show that Algorithm 8 sorts π_n^{prt} with $\lceil n/2 \rceil + 1$ rearrangements. The proofs for n odd and n even are similar, so we will show here only the proof for n odd. The loop invariant for the while loop in line 5 of Algorithm 8 is the following: before the ith iteration, $\pi = ((n{-}5{-}2(i{-}1))\ (n{-}7{-}2(i{-}1)) \ldots 2\ n{-}2\ n{-}1\ n\ n{-}3\ n{-}4 \ldots (n{-}4{-}2(i{-}1))\ (n{-}6{-}2(i{-}1))\ (n{-}8{-}2(i{-}1)) \ldots 1)$. Note that $n{-}5{-}2(i{-}1), n{-}7{-}2(i{-}1), \ldots, 2$ are even numbers in decreasing order, $n{-}3, n{-}4, \ldots, n{-}4{-}2(i{-}1)$ are sorted in decreasing order, and $n{-}6{-}2(i{-}1), n{-}8{-}2(i{-}1), \ldots, 1$ are odd numbers in decreasing order.

It is easy to note that before the first iteration, that is, for $i = 1$, $\pi = (n{-}5\ n{-}7 \ldots 2\ n{-}2\ n{-}1\ n\ n{-}3\ n{-}4\ n{-}6\ n{-}8 \ldots 1)$, because the two first prefix transpositions take π_n^{prt} to it.

During the ith iteration, $\tau_p(2, \pi_{\pi_1 - 1}^{-1})$ is applied, which moves only the first element $n - 5 - 2(i-1)$ and puts it between the odd elements $n{-}4{-}2(i{-}1)$ and $n{-}6{-}2(i{-}1)$ because $\pi_1 - 1 = n{-}5{-}2(i{-}1){-}1 = n{-}6{-}2(i{-}1)$. This transforms the permutation into $((n{-}7{-}2(i{-}1))\ (n{-}9{-}2(i{-}1)) \ldots 2\ (n{-}2)\ (n{-}1)\ n\ (n{-}3)\ (n{-}4) \ldots (n{-}4{-}2(i{-}1))\ (n{-}5{-}2(i{-}1))\ (n{-}6{-}2(i{-}1))\ (n{-}8{-}2(i{-}1)) \ldots 1) = ((n{-}5{-}2i)\ (n{-}7{-}2i) \ldots 2\ (n{-}2)\ (n{-}1)\ n\ (n{-}3)\ (n{-}4) \ldots (n{-}4{-}2i)\ (n{-}6{-}2i)\ (n{-}8{-}2i) \ldots 1)$, which keeps the invariant valid for iteration $i + 1$.

Note that at each iteration, the first element, and only it, is moved. Since the first elements are even numbers in decreasing order, eventually we have $\pi_1 = 2$. Also, since for the first iteration we have $\pi_1 = n{-}5$, the while loop has exactly $\lceil n/2 \rceil - 4$ iterations, which is the number of even elements between $n{-}5$ and 4. Note that after the $(\lceil n/2 \rceil - 4)$th iteration we must have $\pi = (2\ n{-}2\ n{-}1\ n\ n{-}3\ n{-}4\ n{-}5 \ldots 5\ 4\ 3\ 1)$. It is easy to see that the three extra rearrangements in line 7 of Algorithm 8 sort π.

Since there are $\lceil n/2 \rceil - 4$ rearrangements in the while loop, two rearrangements previous to it, and three rearrangements after it, the algorithm uses a total of $\lceil n/2 \rceil + 1$ rearrangements to sort π_n^{prt}, which means $d_{prt}(\pi_n^{prt}) \leq \lceil n/2 \rceil + 1$. □

For SbPSRT, let

$$\pi_n^{psrt} = \begin{cases} (n{-}1\ n{-}3\ n{-}5\ \ldots\ 5\ 3\ 6\ 8\ 10\ \ldots\ n\ 2\ 4\ 1) & \text{if } n \text{ is even} \\ (n\ n{-}2\ n{-}4\ \ldots\ 5\ 3\ 6\ 8\ 10\ \ldots\ n{-}1\ 2\ 4\ 1) & \text{if } n \text{ is odd.} \end{cases} \tag{4.4}$$

Lemma 30. *For $n \geq 6$, $\left\lceil \frac{n-1}{2} \right\rceil \leq d_{psrt}(\pi_n^{psrt}) \leq \left\lceil \frac{n}{2} \right\rceil + 1$.*

Proof. First note that $d_{psrt}(\pi_n^{psrt}) \geq \lceil (n-1)/2 \rceil$ according to Lemma 1 and because $b_{upsrt}(\pi_n^{psrt}) = n - 1$.

Now we show that Algorithm 9 sorts π_n^{psrt} with $\lceil n/2 \rceil + 1$ rearrangements, which shows the upper bound on the distance. Again, the proofs for n odd and n even are similar, so

Algorithm 8 An algorithm to sort π_n^{prt} with prefix reversals and prefix transpositions.

SORT_FAMILY_SBPRT(π, n)

Input: permutation $\pi = \pi_n^{prt}$ and its size $n \geq 7$
Output: number of rearrangements used to sort π

1 **if** $n \bmod 2 \equiv 0$ **then**
2 $\quad \pi \leftarrow \pi \cdot \rho_p(2) \cdot \tau_p(5, \pi_{n-3}^{-1} + 1)$
3 **else**
4 $\quad \pi \leftarrow \pi \cdot \tau_p(3, \pi_{n-4}^{-1}) \cdot \tau_p(2, \pi_n^{-1})$
5 $d \leftarrow 2$
6 **while** $\pi_1 \neq 2$ **do**
7 $\quad \pi \leftarrow \pi \cdot \tau_p(2, \pi_{\pi_1 - 1}^{-1})$
8 $\quad d \leftarrow d + 1$
9 $\pi \leftarrow \pi \cdot \tau_p(\pi_n^{-1} + 1, n + 1) \cdot \rho_p(\pi_2^{-1}) \cdot \rho_p(2)$
10 **return** $d + 3$

we will show here only the proof for n even. The loop invariant for the while loop in line 2 is the following: before the ith iteration, $\pi = ((n{-}1{-}2(i{-}1))\ (n{-}3{-}2(i{-}1))\ \ldots\ 3\ 6\ 8\ \ldots\ (n{-}4{-}2(i{-}1))\ (n{-}2{-}2(i{-}1))\ (n{-}2(i{-}1))\ (n{+}1{-}2(i{-}1))\ \ldots\ n\ 2\ 4\ 1)$.

It is easy to note that, before the first iteration, where $i = 1$, $\pi = (n{-}1\ n{-}3\ n{-}5\ \ldots\ 3\ 6\ 8\ \ldots\ n{-}4\ n{-}2\ n\ 2\ 4\ 1)$, because that is the input permutation π_n^{psrt}.

During the ith iteration, $\tau_p(2, \pi_{\pi_1+1}^{-1})$ is applied, which moves only the first element $n{-}1{-}2(i{-}1)$ and puts it between the even elements $n{-}2{-}2(i{-}1)$ and $n{-}2(i{-}1)$, because $\pi_1{+}1 = n{-}1{-}2(i{-}1){+}1 = n{-}2(i{-}1)$. This transforms the permutation into $((n{-}3{-}2(i{-}1))\ (n{-}5{-}2(i{-}1))\ \ldots\ 3\ 6\ 8\ \ldots\ (n{-}4{-}2(i{-}1))\ (n{-}2{-}2(i{-}1))\ (n{-}1{-}2(i{-}1))\ (n{-}2(i{-}1))\ (n{+}1{-}2(i{-}1))\ \ldots\ n\ 2\ 4\ 1)$, which keeps the invariant valid for iteration $i + 1$.

Note that, for each iteration, the first element, and only it, is moved. Since the first elements of π_n^{psrt} are the odd elements in decreasing order, we eventually have $\pi_1 = 5$. As in the first iteration we have $\pi_1 = n{-}1$, the while loop has exactly $\lceil n/2 \rceil - 3$ iterations, which is the number of odd elements between $n{-}1$ and 7. Note that after the $(\lceil n/2 \rceil{-}3)$th iteration we must have $\pi = (5\ 3\ 6\ 7\ 8\ 9\ \ldots\ n\ 2\ 4\ 1)$. It is easy to see that the four extra rearrangements on line 6 sort it.

Since there are $\lceil n/2 \rceil - 3$ rearrangements on the while loop and four rearrangements after it, the algorithm uses a total of $\lceil n/2 \rceil + 1$ rearrangements to sort π_n^{psrt}, which means $d_{psrt}(\pi_n^{psrt}) \leq \lceil n/2 \rceil + 1$. □

For SBPSR̄, let

$$\pi_n^{spsr} = ((-1^n)n\ (-1^{n-1})(n-1)\ (-1^{n-2})(n-2)\ \ldots\ +2\ -1). \tag{4.5}$$

Lemma 31. *For $n \geq 5$, $n \leq d_{ps\bar{r}}(\pi_n^{spsr}) \leq n + \left\lfloor \frac{n-1}{2} \right\rfloor$.*

Proof. First note that $n{-}1$ is a valid lower bound according to Lemma 2 and because $b_{ps}(\pi_n^{spsr}) = n{-}1$. However, this lower bound is not tight, because if it were, this would imply that a 1-move would always exist. If n is even, then $\pi_n^{spsr} = (+n\ -(n{-}1)\ +(n{-}2)\ -(n{-}3)\ \ldots\ +2\ -1)$ and it is easy to see that there is only one possible 1-move, which will lead to $\pi' = (-n\ -(n{-}1)\ +(n{-}2)\ -(n{-}3)\ \ldots\ +2\ -1)$. On the other hand, it is not possible

Algorithm 9 An algorithm to sort π_n^{psrt} with prefix reversals, prefix transpositions, suffix reversals, and suffix transpositions.

SORT_FAMILY_SBPSRT(π, n)

Input: permutation $\pi = \pi_n^{psrt}$ and its size $n \geq 6$
Output: number of rearrangements used to sort π

1 $d \leftarrow 0$
2 **if** $n \bmod 2 \equiv 0$ **then**
3 **while** $\pi_1 \neq 5$ **do**
4 $\pi \leftarrow \pi \cdot \tau_p(2, \pi^{-1}_{\pi_1+1})$
5 $d \leftarrow d + 1$
6 $\pi \leftarrow \pi \cdot \tau_p(n-1, n) \cdot \tau_p(3, 4) \cdot \tau_p(n-1, n+1) \cdot \rho_p(2)$
7 $d \leftarrow d + 4$
8 **else**
9 $\pi \leftarrow \pi \cdot \tau_p(\pi_3^{-1}, \pi_4^{-1}) \cdot \tau_s(\pi_2^{-1}, \pi_{n-2}^{-1}) \cdot \tau_p(2, \pi_1^{-1})$
10 $d \leftarrow d + 3$
11 **while** $\pi_1 \neq n-1$ **do**
12 $\pi \leftarrow \pi \cdot \tau_p(2, \pi^{-1}_{\pi_1-1})$
13 $d \leftarrow d + 1$
14 $\pi \leftarrow \pi \cdot \rho_p(n-1) \cdot \rho_p(2)$
15 $d \leftarrow d + 2$
16 **return** d

to apply a 1-move in π'. If n is odd, then $\pi_n^{spsr} = (-n\ +(n-1)\ -(n-2)\ +(n-3)\ \ldots\ +2\ -1)$, for which we already cannot find a 1-move. Therefore, the distance is at least n.

Now we show that Algorithm 10 sorts π_n^{spsr} with $n + \lfloor (n-1)/2 \rfloor$ rearrangements. The proofs for n odd and n even are similar, so we will show here only the one for n odd. In the rest of the proof, we consider the strips $x_j = \langle (n-2j+1)\ (n-2j+2) \rangle$, $-x_j = \langle -(n-2j+2)\ -(n-2j+1) \rangle$, $y_j = \langle -(n-2j+2)\ (n-2j+1) \rangle$, and $-y_j = \langle -(n-2j+1)\ (n-2j+2) \rangle$. Let $h = (n-1)/2$. Note that $\pi_n^{spsr} = (y_1\ y_2\ \ldots\ y_h\ -1)$.

First we consider the for loop in line 4. Its loop invariant is the following: if i is odd, then before the ith iteration $\pi = (x_i\ x_{i-2}\ x_{i-4}\ \ldots\ x_1\ -x_2\ -x_4\ -x_6\ \ldots\ -x_{i-1}\ y_{i+1}\ y_{i+2}\ \ldots\ y_h\ -1)$, and if i is even, then before the ith iteration $\pi = (x_i\ x_{i-2}\ x_{i-4}\ \ldots\ x_2\ -x_1\ -x_3\ -x_5\ \ldots\ -x_{i-1}\ y_{i+1}\ y_{i+2}\ \ldots\ y_h\ -1)$.

It is easy to note that before the first iteration, $\pi = (x_1\ y_2\ y_3\ \ldots\ y_h\ -1)$, because the two first rearrangements take π_n^{spsr} to such π.

Now note that the for loop runs from $j = 4$ to $n-1$ with step 2, which means that in the ith iteration, $j = 2i+2$. Also, reversal $\bar{\rho}_p(j)$ involves the j first elements of π, or the $j/2 = i+1$ first strips of size 2. Let i be even. During the ith iteration, reversal $\bar{\rho}_p(j)$ generates $\pi = (-y_{i+1}\ x_{i-1}\ x_{i-3}\ \ldots\ x_2\ -x_1\ -x_3\ \ldots\ -x_i\ y_{i+2}\ y_{i+3}\ \ldots\ y_h\ -1)$ while reversal $\bar{\rho}_p(1)$ generates $\pi = (-x_{i+1}\ x_{i-1}\ x_{i-3}\ \ldots\ x_2\ -x_1\ -x_3\ \ldots\ -x_i\ y_{i+2}\ y_{i+3}\ \ldots\ y_h\ -1)$, which keeps the invariant valid for iteration $i+1$. If i is odd, the argument is similar.

Note that, at each iteration, y_{i+1} is brought to the beginning of the permutation and transformed into x_{i+1}. Also, there are exactly $h-1$ iterations. After the last iteration, we have either $\pi = (x_h\ x_{h-2}\ x_{h-4}\ \ldots\ x_1\ -x_2\ -x_4\ \ldots\ -x_{h-1}\ -1)$ if h is odd or $\pi = (x_h\ x_{h-2}\ x_{h-4}\ \ldots\ x_2\ -x_1\ -x_3\ \ldots\ -x_{h-1}\ -1)$ if h is even. In any case, note that $x_h = \langle 2\ 3 \rangle$.

Now we consider the for loop in line 7 when h is odd; the even case is similar. The

loop invariant in this case is: if i is odd, then before the ith iteration $\pi = (1\ x_h\ x_{h-1}\ \ldots\ x_{h-i+2}\ x_{h-i}\ x_{h-i-2}\ \ldots\ x_2\ -x_1\ -x_3\ \ldots\ -x_{h-i+1})$, and if i is even, then before the ith iteration $\pi = (1\ x_h\ x_{h-1}\ \ldots\ x_{h-i+2}\ x_{h-i}\ x_{h-i-2}\ \ldots\ x_1\ -x_2\ -x_4\ \ldots\ -x_{h-i+1})$.

Before the first iteration, it is easy to note that $\pi = (1\ x_{h-1}\ x_{h-3}\ \ldots\ x_2\ -x_1\ -x_3\ \ldots\ -x_h)$. Now note that the for loop runs from $j = 2$ to $n-1$ with step 2, which means that, in the ith iteration, $j = 2i$. Also, reversal $\bar{\rho}_s(j)$ involves the $n-j+1$ last elements of π, or the $(n-j+1)/2 = h-i-1$ last strips of size 2. Let i be even. During the ith iteration, $\rho_s(i)$ is applied, generating $\pi = (1\ x_h\ x_{h-1}\ \ldots\ x_{h-i+2}\ x_{h-i+1}\ x_{h-i-1}\ x_{h-i-3}\ \ldots\ x_4\ x_2\ -x_1\ -x_3\ \ldots\ -x_{h-i})$, which keeps the invariant valid for iteration $i+1$. If i is odd, the argument is similar.

Note that there are exactly h iterations. Before the last iteration, we have $\pi = (1\ x_h\ x_{h-1}\ x_{h-2}\ \ldots\ x_2\ -x_1)$, and the reversal $\bar{\rho}_s(n-1)$ sorts it.

Since there are $h-1$ iterations in the first loop, each one with two prefix reversals, h iterations in the second loop, each one with one suffix reversal, and two rearrangements out of the loops, the algorithm uses a total of $2h-2+h+3 = 2(n-1)/2+1+(n-1)/2 = n+(n-1)/2$ rearrangements to sort π_n^{spsr}, which means $d_{ps\bar{r}}(\pi_n^{spsr}) \leq n+\lfloor (n-1)/2 \rfloor$. □

Algorithm 10 An algorithm to sort π_n^{spsr} with signed prefix reversals and signed suffix reversals.

SORT_FAMILY_SBPSR̄(π, n)
Input: permutation $\pi = \pi_n^{spsr}$ and its size $n \geq 5$
Output: number of rearrangements used to sort π
1 $d \leftarrow 0$
2 **if** $n \bmod 2 = 0$ **then**
3 $\quad \pi \leftarrow \pi \cdot \bar{\rho}_p(1)$
4 $\quad d \leftarrow 1$
5 $\pi \leftarrow \pi \cdot \bar{\rho}_p(3-(n \bmod 2)) \cdot \bar{\rho}_p(1)$
6 $d \leftarrow d+2$
7 **for** $j \leftarrow 5-(n \bmod 2)$ **to** $n-1$ **by** 2 **do**
8 $\quad \pi \leftarrow \pi \cdot \bar{\rho}_p(j) \cdot \bar{\rho}_p(1)$
9 $\quad d \leftarrow d+1$
10 $\pi \leftarrow \pi \cdot \bar{\rho}_p(n)$
11 $d \leftarrow d+1$
12 **for** $j \leftarrow 2$ **to** $n-1-(1-(n \bmod 2))$ **by** 2 **do**
13 $\quad \pi \leftarrow \pi \cdot \bar{\rho}_s(j)$
14 $\quad d \leftarrow d+1$
15 **return** d

For SBPR̄T, let

$$\pi_n^{sprt} = (+4\ +3\ +2\ +1\ -5\ -6\ -7\ \ldots\ -(n-1)\ -n). \tag{4.6}$$

Lemma 32. *For* $n \geq 5$, $\lceil \frac{n}{2} \rceil + 1 \leq d_{p\bar{r}t}(\pi_n^{sprt}) \leq n+2-\lfloor \frac{n}{4} \rfloor$.

Proof. First note that $\lceil n/2 \rceil$ is a valid lower bound according to Lemma 2 and because $b_p(\pi_n^{sprt}) = n$, but it is not a tight lower bound. If it was, then when n is even the sorting should only have 2-moves, but it is easy to see that this is not possible. When n is odd,

the sorting would need to have $\lceil n/2 \rceil - 1$ 2-moves and one 1-move. Applying a 2-move in π_n^{sprt} is not possible, and the only 1-move that exists places 4 after 3. However, now it is not possible to find a 2-move either. Therefore, the distance is at least $\lceil n/2 \rceil + 1$.

There is a simple algorithm that sorts π_n^{sprt}: it reverts the segment $+4, +3, +2, +1$ generating $(-1\ -2\ -3\ \ldots\ -n)$, reverts the whole permutation, and then uses the algorithm presented by Dias and Meidanis [24], which uses at most $n - \lfloor n/4 \rfloor$ prefix transpositions to sort $(n\ n{-}1\ \ldots\ 1)$. Therefore, it uses at most $n + 2 - \lfloor n/4 \rfloor$ rearrangements to sort π_n^{sprt}, and $d_{p\bar{r}t}(\pi_n^{sprt}) \leq n + 2 - \lfloor n/4 \rfloor$. □

For SbPS$\bar{\text{R}}$T, let

$$\pi_n^{spsrt} = (-1\ \ +2\ \ -3\ \ +4\ \ \ldots\ \ -(n-1)\ \ +n). \tag{4.7}$$

Lemma 33. *For $n \geq 6$ and even, $\frac{n}{2} \leq d_{ps\bar{r}t}(\pi_n^{spsrt}) \leq n$.*

Proof. First note that the lower bound is valid because $d_{ps\bar{r}t}(\pi) \geq b_{psrt}(\pi)/2$ for all π, according to Lemma 2, and because $b_{psrt}(\pi_n^{spsrt}) = n - 1$.

Now we show that Algorithm 11 sorts π_n^{spsrt} with n rearrangements. The loop invariant for the for loop is: before the ith iteration, $\pi = (-(2i{-}1)\ (2i)\ -(2i{+}1)\ (2i{+}2)\ \ldots\ -(n{-}1)\ n\ 1\ 2\ 3\ \ldots\ (2i{-}2))$.

It is easy to see that before the first iteration the permutation is of the claimed format because no rearrangement was applied yet. During the ith rearrangement, we have $\bar{\rho}_p(1)$ applied, generating $((2i{-}1)\ (2i)\ -(2i{+}1)\ (2i{+}2)\ \ldots\ -(n{-}1)\ n\ 1\ 2\ 3\ \ldots\ (2i{-}2))$, and then $\tau_p(3, n+1)$, generating $(-(2i{+}1)\ (2i{+}2)\ -(2i{+}3)\ (2i{+}4)\ \ldots\ -(n{-}1)\ n\ 1\ 2\ 3\ \ldots\ (2i{-}2)\ (2i{-}1)\ (2i))$, which keeps the invariant valid for iteration $i{+}1$.

Note that the for loop has exactly $n/2$ iterations (one for each odd number between 1 and $n{-}1$). Before the $(n/2)$th iteration, we must have $\pi = (-(n{-}1)\ n\ 1\ 2\ 3\ \ \ldots\ \ (n{-}2))$, and the two rearrangements inside the iteration sort it.

Since there are $n/2$ iterations and each of them has two rearrangements, the algorithm uses a total of n rearrangements to sort π_n^{spsrt}, which means $d_{ps\bar{r}t}(\pi_n^{spsrt}) \leq n$. □

Algorithm 11 An algorithm to sort π_n^{spsrt} with signed prefix reversals, prefix transpositions, signed suffix reversals, and suffix transpositions when n is even.

SORT_FAMILY_SbPSR̄T_EVEN(π, n)

Input: permutation $\pi = \pi_n^{spsrt}$ and its size $n \geq 8$
Output: number of rearrangements used to sort π

```
1  d ← 0
2  for i ← 1 to n − 1 by 2 do
3      π ← π · ρ̄_p(1) · τ_p(3, n + 1)
4      d ← d + 1
5  return d
```

Lemma 34. *For $n \geq 5$ and odd, $\frac{n-1}{2} \leq d_{ps\bar{r}t}(\pi_n^{spsr}) \leq n$.*

Proof. First note that the lower bound is valid because $d_{ps\bar{r}t}(\pi) \geq b_{psrt}(\pi)/2$ for all π, according to Lemma 2, and because $b_{psrt}(\pi_n^{spsr}) = n - 1$.

Now we show that Algorithm 12 sorts π_n^{spsr} with n rearrangements. The loop invariant for the for loop is: before the ith iteration, $\pi = ((n–5–2(i–1)) –(n–6–2(i–1)) (n–7–2(i–1)) –(n–8–2(i–1)) \dots 2 –1 –n –(n–1) –(n–2) (n–3) –(n–4)) –(n–5) \dots –(n–4–2(i–1)))$. Note that $–(n–4), –(n–5), \dots, –(n–4–2(i–1))$ only contains negative elements.

It is easy to see that before the first iteration $\pi = ((n–5) –(n–6) (n–7) –(n–8) \dots 2 –1 –n –(n–1) –(n–2) (n–3) –(n–4))$, because the three first rearrangements take π_n^{spsr} to it.

During the ith iteration, $\bar{\rho}_p(1)$ is applied, generating $(–(n–5–2(i–1)) –(n–6–2(i–1)) (n–7–2(i–1)) –(n–8–2(i–1)) \dots 2 –1 –n –(n–1) –(n–2) (n–3) –(n–4) –(n–5) \dots –(n–4–2(i–1)))$, and then $\tau_p(3, n+1)$ is applied, generating $((n–7–2(i–1)) –(n–8–2(i–1))) \dots 2 –1 –n –(n–1) –(n–2) (n–3) –(n–4) –(n–5) \dots –(n–4–2(i–1)) –(n–5–2(i–1)) –(n–6–2(i–1)))$. This means that the invariant is valid for iteration $i+1$.

Note that the for loop has exactly $(n-1)/2-3$ iterations (one for each even number between 4 and $n–5$). Before the $((n-1)/2-3)$th iteration, we must have $\pi = (4 –3 2 –1 –n –(n–1) –(n–2) (n–3) –(n–4) –(n–5) \dots –6 –5)$, and the four rearrangements of line 4 sort it.

Since there are $(n-1)/2-3$ iterations, each of which has two rearrangements, three rearrangements previous to it, and four rearrangements after it, the algorithm uses a total of $2(n-1)/2-6+7 = n$ rearrangements to sort π_n^{spsr}, which means $d_{ps\bar{r}t}(\pi_n^{spsr}) \leq n$. □

Algorithm 12 An algorithm to sort π_n^{spsr} with signed prefix reversals, prefix transpositions, signed suffix reversals, and suffix transpositions when n is odd.

SORT_FAMILY_SBPSR̄T_ODD(π, n)

Input: permutation $\pi = \pi_n^{spsr}$ and its size $n \geq 7$
Output: number of rearrangements used to sort π

1 $\pi \leftarrow \pi \cdot \tau_p(2, n+1) \cdot \bar{\rho}_p(1) \cdot \tau_p(\pi_{n-5}^{-1}, n+1)$
2 $d \leftarrow 3$
3 **for** $i \leftarrow n-5$ **downto** 4 **by** -2 **do**
4 $\quad \pi \leftarrow \pi \cdot \bar{\rho}_p(1) \cdot \tau_p(3, n+1)$
5 $\quad d \leftarrow d+1$
6 $\pi \leftarrow \pi \cdot \bar{\rho}_p(1) \cdot \tau_p(\pi_{n-3}^{-1}+1, n+1) \cdot \bar{\rho}_p(n-1) \cdot \tau_p(\pi_1^{-1}, n+1)$
7 **return** $d+4$

The next lemmas give lower and upper bounds for the diameters of the problems that involve prefix and suffix rearrangements addressed by us, considering the presented families and the already existing bounds for problems that involve only prefix rearrangements.

Lemma 35. *For $n \geq 8$, $D_{psr}(n) \geq n-1$ and for $n \geq 1$, $D_{psr}(n) \leq \frac{18n}{11} + O(1)$.*

Proof. The lower bound is valid due to family π_n^{psr}, as Lemma 27 shows. The upper bound is valid because $D_{psr}(n) \leq D_{pr}(n)$, since $d_{psr}(\pi) \leq d_{pr}(\pi)$ for any π, and $D_{pr}(n) \leq 18n/11 + O(1)$ [15]. □

Lemma 36. *For $n \geq 3$, $\left\lceil \frac{n-1}{2} \right\rceil + 1 \leq D_{pst}(n) \leq n - \log_{7/2} n$.*

Proof. The lower bound is valid due to family π_n^{pst}, as Lemma 28 shows. The upper bound is valid because $D_{pst}(n) \leq D_{pt}(n)$, since $d_{pst}(\pi) \leq d_{pt}(\pi)$ for any π, and $D_{pt}(n) \leq n - \log_{7/2} n$ [13]. □

Lemma 37. *For $n \geq 7$, $D_{prt}(n) \geq \lceil \frac{n}{2} \rceil$ and for $n \geq 1$, $D_{prt}(n) \leq n - \log_{7/2} n$.*

Proof. The lower bound is valid due to family π_n^{prt}, as Lemma 29 shows. The upper bound is valid because $D_{prt}(n) \leq \min\{D_{pr}(n), D_{pt}(n)\}$, since $d_{prt}(\pi) \leq \min\{d_{pr}(\pi), d_{pt}(\pi)\}$ for any π, $D_{pr}(n) \leq 18n/11 + O(1)$ [15], and $D_{pt}(n) \leq n - \log_{7/2} n$ [13]. □

Lemma 38. *For $n \geq 6$, $D_{psrt}(n) \geq \lceil \frac{n-1}{2} \rceil$ and for $n \geq 1$, $D_{psrt}(n) \leq n - \log_{7/2} n$.*

Proof. The lower bound is valid due to family π_n^{psrt}, as Lemma 30 shows. The upper bound is valid because $D_{psrt}(n) \leq \min\{D_{psr}(n), d_{pst}(n)\} \leq \min\{D_{pr}(n), D_{pt}(n)\}$, since $d_{psrt}(\pi) \leq \min\{d_{psr}(\pi), d_{pst}(\pi)\} \leq \min\{d_{pr}(\pi), d_{pt}(\pi)\}$ for any π, $D_{pr}(n) \leq 18n/11 + O(1)$ [15], and $D_{pt}(n) \leq n - \log_{7/2} n$ [13]. □

Lemma 39. *For $n \geq 5$, $D_{ps\bar{r}}(n) \geq n$ and for $n \geq 16$, $D_{ps\bar{r}}(n) \leq 2n - 6$.*

Proof. The lower bound is valid because of family π_n^{spsr}, as Lemma 31 shows. The upper bound is valid because $D_{ps\bar{r}}(n) \leq D_{p\bar{r}}(n)$, since $d_{ps\bar{r}}(\pi) \leq d_{p\bar{r}}(\pi)$ for any π, and $D_{p\bar{r}}(n) \leq 2n - 6$ [17]. This is true because any sorting sequence for SBPR̄ is valid for SBPSR̄. □

Lemma 40. *For $n \geq 5$, $D_{p\bar{r}t}(n) \geq \lceil \frac{n}{2} \rceil + 1$ and for $n \geq 16$, $D_{p\bar{r}t}(n) \leq 2n - 6$.*

Proof. The lower bound is valid because of family π_n^{sprt}, as Lemma 32 shows. The upper bound is valid because $D_{p\bar{r}t}(n) \leq D_{p\bar{r}}(n)$, since $d_{p\bar{r}t}(\pi) \leq d_{p\bar{r}}(\pi)$ for any π, and $D_{p\bar{r}}(n) \leq 2n - 6$ [17]. Note that a sorting sequence for SBPT is not always valid for this problem, since it deals only with unsigned permutations. □

Lemma 41. *For $n \geq 5$, $D_{ps\bar{r}t}(n) \geq \lceil \frac{n-1}{2} \rceil$ and for $n \geq 1$, $D_{ps\bar{r}t}(n) \leq 2n - 6$.*

Proof. The lower bound is valid because of families π_n^{sprt} and π_n^{spsr}, as Lemmas 33 and 34 show. The upper bound is true because $D_{ps\bar{r}t}(n) \leq D_{ps\bar{r}}(n) \leq D_{p\bar{r}}(n)$, since $d_{ps\bar{r}t}(\pi) \leq d_{ps\bar{r}}(\pi) \leq d_{p\bar{r}}(\pi)$ for any π, and $D_{p\bar{r}}(n) \leq 2n - 6$ [17]. □

Table 4.5 shows some exact values for the diameter of the studied problems. It has been obtained from the results given by the Rearrangement Distance Database tool[2] of Galvão and Dias [31]. From this table we can see that $D_{psr}(n) = n$ for $7 \leq n \leq 13$, $D_{pst}(n) = d_{pst}(\pi_n^{pst} = \eta_n)$ for $1 \leq n \leq 12$, $D_{psrt}(n) = \lceil n/2 \rceil + 1$ for $6 \leq n \leq 13$, $D_{ps\bar{r}}(n) = n + \lfloor (n-1)/2 \rfloor$ for $5 \leq n \leq 10$, $D_{p\bar{r}t}(n) = d_{p\bar{r}t}(\pi_n^{sprt})$ for $2 \leq n \leq 10$, and $D_{ps\bar{r}t}(n) = d_{ps\bar{r}t}(\pi_n^{spsrt})$ for $n \in \{8, 10\}$ and $D_{ps\bar{r}t}(n) = d_{ps\bar{r}t}(\pi_n^{spsr})$ for $n \in \{7, 9\}$. We can also show that $d_{psr}(\pi_n^{psr}) = n$ for $8 \leq n \leq 15$ and $d_{ps\bar{r}}(\pi_n^{spsr}) = n + \lfloor (n-1)/2 \rfloor$ for $5 \leq n \leq 12$.

It is worth noticing that, for $n = 7$, the only two permutations for which $d_{p\bar{r}t}(\pi) = D_{p\bar{r}t}(7) = 8$ are $\pi = \pi_7^{sprt}$ and $\pi = (+3\ +2\ +1\ -4\ -5\ -6\ -7)$, and the only two permutations for which $d_{psr}(\pi) = D_{psr}(7) = 7$ are $\pi = (7\ 3\ 5\ 2\ 6\ 4\ 1)$ and $\pi = (7\ 4\ 2\ 6\ 3\ 5\ 1)$.

The next conjectures are directly based on the results mentioned above.

Conjecture 1. *For $n \geq 8$, $D_{psr}(n) = d_{psr}(\pi_n^{psr}) = n$.*

[2] Available at *http://mirza.ic.unicamp.br:8080/bioinfo*.

Table 4.5: Diameter values for small values of n.

n	$D_{prt}(n)$	$D_{psr}(n)$	$D_{pst}(n)$	$D_{psrt}(n)$	$D_{p\bar{r}t}(n)$	$D_{ps\bar{r}}(n)$	$D_{ps\bar{r}t}(n)$
1	0	0	0	0	1	1	1
2	1	1	1	1	3	3	2
3	2	2	2	1	4	4	3
4	2	3	3	2	5	6	4
5	3	4	3	3	6	7	5
6	4	5	4	4	7	8	6
7	5	7	5	5	8	10	7
8	5	8	6	5	8	11	7
9	6	9	6	6	9	13	8
10	7	10	7	6	10	14	9
11	7	11	8	7	-	-	-
12	8	12	8	7	-	-	-
13	9	13	9	8	-	-	-

Conjecture 2. *For $n \geq 1$, $D_{pst}(n) = d_{pst}(\eta_n) = n - \left\lfloor \frac{n}{4} \right\rfloor$.*

Conjecture 3. *For $n \geq 6$, $D_{psrt}(n) = d_{psrt}(\pi_n^{psrt}) = \left\lceil \frac{n}{2} \right\rceil + 1$.*

Conjecture 4. *For $n \geq 5$, $D_{p\bar{r}t}(n) = d_{p\bar{r}t}(\pi_n^{sprt})$.*

Conjecture 5. *For $n \geq 5$, $D_{ps\bar{r}}(n) = d_{ps\bar{r}}(\pi_n^{spsr}) = n + \left\lfloor \frac{n-1}{2} \right\rfloor$.*

Conjecture 6. *For $n \geq 8$ and n even, $D_{ps\bar{r}t}(n) = d_{ps\bar{r}t}(\pi_n^{spsrt})$. For $n \geq 7$ and n odd, $D_{ps\bar{r}t}(n) = d_{ps\bar{r}t}(\pi_n^{spsr})$.*

4.9 Summary of the Chapter

Table 4.6 summarizes the best approximation factors and the bounds for the diameters of the problems that were studied in this chapter.

Table 4.6: Summary of the results obtained for prefix and suffix rearrangement problems.

Rearrangements	**Approx. Factor**	**Diameter**	
		Lower Bound	**Upper Bound**
Pref. and Suf. Reversals	2 (Thm. 1)	$n-1$ (Lem. 35)	$\frac{18n}{11} + O(1)$ (Lem. 35)
Pref. and Suf. Transpositions	2 (Thm 2)	$\left\lceil \frac{n-1}{2} \right\rceil + 1$ (Lem. 36)	$n - \log_{7/2} n$ (Lem. 36)
Pref. Reversals and Transpositions	$2 + \frac{4}{b_{upr}(\pi)}$ [23]	$\left\lceil \frac{n}{2} \right\rceil$ (Lem. 37)	$n - \log_{7/2} n$ (Lem. 37)
Pref. and Suf. Reversals and Transp.	$2 + \frac{4}{b_{upsrt}(\pi)}$ (Thm 3)	$\left\lceil \frac{n-1}{2} \right\rceil$ (Lem. 38)	$n - \log_{7/2} n$ (Lem. 38)
Sig. Pref. and Suf. Reversals	2 (Thm 4)	n (Lem. 39)	$2n - 6$ (Lem. 39)
Sig. Pref. Reversals and Transpositions	$2 + \frac{4}{b_p(\pi)}$ (Thm 5)	$\left\lceil \frac{n}{2} \right\rceil + 1$ (Lem. 40)	$2n - 6$ (Lem. 40)
Sig. Pref. and Suf. Reversals and Transp.	$2 + \frac{4}{b_{upr}(\pi)}$ (Thm 6)	$\left\lceil \frac{n-1}{2} \right\rceil$ (Lem. 41)	$2n - 6$ (Lem. 41)

Chapter 5

Known Results for Length-Weighted Approach

This chapter presents a review of the existing results, to the best of our knowledge, for the length-weighted approach, where the cost of sorting a permutation is calculated based on the length of the rearrangements involved in the process (specifically, $f(\ell) = \ell^\alpha$ with $\alpha > 0$). We consider related work only, namely those in which the rearrangement model involves general or restricted versions of reversals and/or transpositions.

Pinter and Skiena [50] gave the first results on length-weighted rearrangements. They presented a $O(\lg^2 n)$-approximation algorithm for the problem of Sorting by Length-Weighted Reversals (SBWR) for $\alpha = 1$. They also gave a $O(n \lg^2 n)$ upper bound on the diameter for this variation.

Bender *et al.* [5] then presented an analysis for several values of $\alpha > 0$, also for SBWR. They started relating the sorting of a binary string with the sorting of a permutation and, therefore, also gave results for Sorting Binary Strings by Length-Weighted Reversals (SBBWR). All of their results are given in Table 5.1, but we highlight here that, for SBWR and $\alpha = 1$, they guaranteed an approximation factor of $O(\lg n)$ and they showed a lower bound on the diameter of $\Omega(n \lg n)$. Bender *et al.* [5] also showed that, for $\alpha \geq 3$, SBWR becomes polynomially solvable because it suffices to use Bubble Sort to solve it. In general, they showed that reversals of length greater than 2 can be replaced by reversals of length equal to 2 while decreasing the sorting cost and that Bubble Sort is optimal among all algorithms that use only reversals of length 2.

Swidan *et al.* [54] considered Sorting by Signed Length-Weighted Reversals (SBW$\bar{\text{R}}$), also relating it to the sorting of binary strings by considering the problem of Sorting Binary Strings by Signed Length-Weighted Reversals (SBBW$\bar{\text{R}}$). All of their results are also given in Table 5.1 and we highlight that they were able to guarantee an approximation factor of $O(\lg n)$ when $\alpha = 1$ for SBW$\bar{\text{R}}$.

To our knowledge, there is no result regarding Sorting by Length-Weighted Transpositions (SBWT) or Sorting by Length-Weighted Reversals and Transpositions (SBWRT). However, we point out that both the $O(\lg^2 n)$-approximation presented by Pinter and Skiena [50] and the $O(\lg n)$-approximation presented by Bender *et al.* [5] can be easily adapted to use only transpositions or reversals and transpositions. More than that, the algorithm of Pinter and Skiena [50] would have the same approximation factor for both

SBWT and SBWRT because the analysis would be identical.

Nguyen *et al.* [49] considered a slightly different variant of SBBWR and SBWR. Given an integer k, no reversal can have length greater than k. All of their results are summarized in Table 5.1 as well.

5.1 Summary of the Chapter

Table 5.1 summarizes the best-known approximation factors and the best-known bounds for the diameters of the problems that were mentioned throughout this chapter.

Table 5.1: Summary of best-known results for length-weighted genome rearrangement problems. A '-' indicates that there is no known result.

Rearrangements	**Parameter**	**Best Approx. Factor**		**Diameter**		
		Bin. Str.	**Perm.**	**Bin. Str.**	**Perm. (Lower)**	**Perm. (Upper)**
Reversals [5]	$0 < \alpha < 1$	$O(1)$	-	$\theta(n)$	$\Omega(n)$	$O(n \lg n)$
	$\alpha = 1$	1	$O(\lg n)$	$\theta(n \lg n)$	$\Omega(n \lg n)$	$O(n \lg^2 n)$
	$1 < \alpha < 2$	$O(1)$	$O(\lg n)$	$\theta(n^\alpha)$	$\theta(n^\alpha)$	
	$2 \leq \alpha < 3$	1	2	$\theta(n^2)$	$\theta(n^2)$	
	$\alpha \geq 3$		1			
Sig. Reversals [54]	$0 < \alpha < 1$	$O(1)$	-	$\theta(n)$	$\Omega(n)$	$O(n \lg n)$
	$\alpha = 1$	3	$O(\lg n)$	$\theta(n \lg n)$	$\Omega(n \lg n)$	$O(n \lg^2 n)$
	$1 < \alpha < 2$	$O(1)$	$O(\lg n)$	$\theta(n^\alpha)$	$\theta(n^\alpha)$	
	$\alpha \geq 2$	$O(1)$	$O(1)$	$\theta(n^2)$	$\theta(n^2)$	
Reversals of length at most k [49]	$0 < \alpha < 1$	-	-	$\theta(n + n^2 k^{\alpha-2})$	$\Omega(n + n^2 k^{\alpha-2})$	$O(n \log n + n^2 k^{\alpha-2})$
	$\alpha = 1$	-	-	$\theta(n \log k + \frac{n^2}{k})$	$\Omega(n \log n + \frac{n^2}{k})$	$O(n \log n \log k + \frac{n^2}{k})$
	$1 < \alpha < 2$	-	-	$\theta(\frac{n^2}{k^{\alpha-2}})$	$\Omega(n^2 k^{\alpha-2})$	
	$\alpha \geq 2$	-	-	$\theta(n^2)$	$\theta(n^2)$	
	$k = \Omega(n)$	$O(1)$	$O(\log n)$	-	-	-
	$k = o(n)$	$2 \lg n + 1$	$2 \lg^2 n + \lg n$	-	-	-

Chapter 6

Results Obtained for Length-Weighted Approach

This chapter also considers the length-weighted approach, where the sorting distance of a permutation is calculated based on the length of the rearrangements involved in the process, but it presents the results that we obtained. If necessary, review the notation and definitions on binary strings given in Section 2.2.

Whenever dealing with sorting by length-weighted rearrangement problems, we consider that $\bar{n}$ is the size of a permutation π or of a binary string T (that is, their total number of elements) and that n is their number of *valid* elements, which are defined next. The valid elements are those which can be affected by the rearrangements, i.e, it is valid to apply rearrangements over such elements.

Definition 21. *When only prefix rearrangements are being considered, the number n of* valid *elements in a permutation π is the maximum value i, for $0 \leq i \leq \bar{n}$, such that $\pi_i \neq i$. The number n of valid elements in a binary string T either is the maximum value i, for $1 \leq i \leq \bar{n}$, such that $t_i \neq 1$, or is 0, if $T = 0^k 1^{\bar{n}-k}$ for some $k \geq 0$.*

Definition 22. *When prefix and suffix rearrangements are being considered, let positions i and j be such that (i) $0 \leq i < j \leq \bar{n}+1$, (ii) for all $\ell \in [i+1..j-1]$ we have $\pi_\ell = \ell$, (iii) for all $\ell \in [1..i]$ we have $\pi_\ell \leq i$, (iv) for all $\ell \in [j..\bar{n}]$ we have $\pi_\ell \geq j$, (v) $j-i$ is maximum, and (vi) if $j = i+1$ then $i \neq 0$ and $i \neq \bar{n}$. If positions i and j exist, then the number of* valid *elements in a permutation π is $n = i + \bar{n} - j + 1$ and we say that the permutation is* separated by i and j*. If they do not exist, then $n = \bar{n}$ and the permutation is not separated.*

For prefix and suffix rearrangements, note that $n = \bar{n}$ can happen even if the permutation is separated by i and j. Also note that condition (vi) exists because otherwise every permutation would be separated.

Definition 23. *When general rearrangements are being considered, the number of* valid *elements in a permutation π (resp. in a binary string T) is the maximum value $n = j-i-1$, for $0 \leq i < j \leq \bar{n}+1$, such that $\pi_k = k$ for $0 \leq k \leq i$ and for $j \leq k \leq \bar{n}+1$ (resp. $t_k = 0$ for $0 \leq k \leq i$ and $t_k = 1$ for $j \leq k \leq \bar{n}+1$).*

Note that, by definition, the number of valid elements in the sorted binary string and in the identity permutation is zero. Also note that, even if $n = \bar{n}$ in a permutation, we can have other elements in the correct position, that is, $\pi_t = t$ for some t.

Example 16. *Let* $\pi = (1\ 2\ 4\ 3\ 5\ 6\ 7\ 9\ 8\ 10\ 11)$. *We have that* $\bar{n} = 11$ *but for prefix rearrangements* $n = 9$ *(despite the fact that* 1, 2, 5, 6, *and* 7 *are in their correct position), for prefix and suffix rearrangements* $n = 8$ *(because* 5, 6, *and* 7 *are in their correct position, there is no other such strip with greater size, all elements less than* 5 *are before it, and all elements greater than* 7 *are after it), and for general rearrangements* $n = 7$ *(because of* 1, 2, 10, *and* 11*).*

Example 17. *Consider prefix and suffix rearrangements. Permutation* (1 2 4 3 5 6 7 9 8 10 11)*, of the previous example, is separated by positions* 4 *and* 8 *because segment* $5, 6, 7$ *is in its correct position and is the biggest one with such feature. Permutation* (5 1 4 3 2 7 6 9 8 15 13 14 12 10 11)*, on the other hand, does not have any element in its correct position. However, by Definition 22, it is separated by positions* 5 *and* 6 *(also by positions* 7 *and* 8*, as well as* 9 *and* 10*). Note that in this case the number of valid elements is* $n = \bar{n} = 15$. *Permutation* (6 4 3 1 2 5) *is not separated by any pair of positions, even though element* 3 *is in its correct position, so* $n = \bar{n} = 6$.

The intuition between our definition of valid elements comes from the fact that, when $\alpha > 0$ and when prefix or general rearrangements are being considered, we can act only over the valid elements, as Lemma 42 shows. Basically, the other elements are already in their correct positions, thus there is no need to involve them in the sorting process. Note that this is not necessarily true when we consider prefix and suffix rearrangements, that is, if the permutation is separated by some i and j, an optimal solution may have to involve elements between $i + 1$ and $j - 1$ (specially if $j - i$ is small and i or j are close to 1 or to $\bar{n}$, respectively). Despite that, the definition of valid elements for this case will be useful afterwards.

Lemma 42. *Let* $\alpha > 0$ *and let* π *(resp.* T*) be a permutation (resp. a binary string) with* $\bar{n}$ *elements. Considering prefix rearrangements, any optimal sorting sequence for* π *(resp. for* T*) does not affect elements between* $n + 1$ *and* $\bar{n}$*, where* n *is as in Definition 21. Considering general rearrangements, any optimal sorting sequence for* π *(resp. for* T*) does not affect elements between* 1 *and* i *and between* j *and* $\bar{n}$*, where* i *and* j *are as in Definition 23.*

Proof. In order to prove the result for the prefix case, let $\mathcal{S} = (\lambda_1, \lambda_2, \ldots, \lambda_q)$ be a sequence of prefix rearrangements that optimally sorts π (resp. T) where $\lambda_k \in \{\rho_p, \tau_p\}$ for $1 \leq k \leq q$. Let λ_ℓ be a prefix rearrangement of this sequence that affects a segment that contains a subsegment s of elements from $n + 1$ to $\bar{n}$. Exclude s from λ_ℓ and from all subsequent rearrangements. This modified sequence also sorts π (resp. T) but with a smaller cost, which is a contradiction to the fact that the original sequence was optimal.

A similar analysis can be done to prove the case for general rearrangements. □

Now we start by giving some lower bounds on the distance of permutations and binary strings. For prefix rearrangements, there exists a trivial lower bound for both permuta-

tions and binary strings, as Lemma 43 shows. For prefix and suffix rearrangements, there exists a similar result for $\alpha = 1$, as Lemma 44 shows.

Lemma 43. *Let $\beta \in \{pr, pt, prt\}$ and $\alpha > 0$. For any unsigned permutation π and unsigned binary string T with n valid elements,*

$$c_\beta^\alpha(T) \geq n^\alpha \text{ and } d_\beta^\alpha(\pi) \geq n^\alpha.$$

Proof. For binary strings, since $t_n \neq 1$ by Definition 21, at some point one rearrangement of length n will have to be performed in order to sort the binary string, which will cost $f(n) = n^\alpha$.

Likewise, for permutations, at least one rearrangement that places the element n in its right position should be done, which will also cost $f(n) = n^\alpha$. □

Lemma 44. *Let $\beta \in \{psr, pst, psrt\}$ and $\alpha = 1$. For any unsigned permutation π with n valid elements and $\bar{n}$ elements,*

$$d_\beta^\alpha(\pi) \geq n.$$

Proof. If π is not separated, then $n = \bar{n}$ and every element of the permutation will be affected by at least one rearrangement at some point (to put the element in its correct position), which would contribute with a cost of one for each element; therefore, at least a total cost of n will be necessary. This is true because either π does not have any element such that $\pi_k = k$ (therefore no element is in its correct position) or π has at least one element k such that $\pi_k = k$ and another element ℓ such that $\pi_\ell^{-1} < k$ but $\ell > k$ (or $\pi_\ell^{-1} > k$ but $\ell < k$). Note that such ℓ must exist, otherwise the permutation would be separated. In this last case, the rearrangement(s) needed to move ℓ to its correct position would involve k at some point, which means that, indeed, every element will be affected at least once.

If π is separated by positions i and j (that is, $n = i + \bar{n} - j + 1$), then each element between 1 and i and between j and $\bar{n}$ will be affected by at least one rearrangement at some point so that it can be put in its correct position (an argument similar to the one given above shows this). Therefore, each will contribute with a cost of at least one unit, which means that at least a cost of $(i) + (\bar{n} - j - 1) = n$ will be necessary. □

Given any binary string $T = 0^{w_0}1^{w_1} \dots 0^{w_{2g}}1^{w_{2g+1}}$, we define $P(T) = (\sum_{i=0}^{2g} w_i^\alpha)/4$. Lemma 45 shows a lower bound of $P(T)$ for $c_\beta^\alpha(T)$ when $\beta \in \{pr, pt, prt\}$ and when $0 < \alpha < 1$.

Lemma 45. *Let $\beta \in \{pr, pt, prt\}$ and $0 < \alpha < 1$. For any unsigned binary string $T = 0^{w_0}1^{w_1} \dots 0^{w_{2g}}1^{w_{2g+1}}$ that is not sorted,*

$$c_\beta^\alpha(T) \geq P(T).$$

Proof. To show that $c_{pt}^\alpha(T) \geq P(T)$, we will prove by induction on q that if a solution uses exactly q prefix transpositions, then the cost is at least $P(T)$.

For the base case, $q = 1$, the original sequence must be $0^{w_0}1^{w_1}0^{w_2}1^{w_3}$. The minimum cost to finish the sorting with a prefix transposition is $(w_0 + w_1 + w_2)^\alpha$ (it will exchange the

first two blocks with the third one). When $0 < \alpha < 1$, it is valid that $(a+b)^\alpha \geq (a^\alpha + b^\alpha)/2$. Therefore,

$$c^\alpha_{pt}(T) = (w_0+w_1+w_2)^\alpha \geq \frac{((w_0+w_1)^\alpha + w_2^\alpha)}{2} \geq \frac{\left(\frac{(w_0^\alpha+w_1^\alpha)}{2} + w_2^\alpha\right)}{2} \geq \frac{(w_0^\alpha + w_1^\alpha + w_2^\alpha)}{4} = P(T).$$

Suppose now that for all integers less than or equal to q the claim holds. So, for $q+1$, assume that the first prefix transposition τ_p of an optimal solution has length ℓ and transforms T into $T \cdot \tau_p$. Thus, $c^\alpha_{pt}(T) = c^\alpha_{pt}(T \cdot \tau_p) + \ell^\alpha$. Also, by the induction hypothesis, $c^\alpha_{pt}(T \cdot \tau_p) \geq P(T \cdot \tau_p)$. Therefore,

$$c^\alpha_{pt}(T) = c^\alpha_{pt}(T \cdot \tau_p) + \ell^\alpha \geq P(T \cdot \tau_p) + \ell^\alpha.$$

It suffices then to show that $P(T \cdot \tau_p) + \ell^\alpha \geq P(T)$ to finish the proof. We will show, equivalently, that $P(T) - P(T \cdot \tau_p) \leq \ell^\alpha$.

Suppose that $\tau_p = \tau_p(i, j)$. As depicted in Equation (6.1), let x be the maximum position such that $t_1 = t_h$ for all $1 \leq h \leq x$, y be the minimum position such that $t_{i-1} = t_h$ for all $y \leq h \leq i-1$, z be the maximum position such that $t_i = t_h$ for all $i \leq h \leq z$, w be the minimum position such that $t_{j-1} = t_h$ for all $w \leq h \leq j-1$, and k be the maximum position such that $t_j = t_h$ for all $j \leq h \leq k$. So, let a, b, c, d, and e be the amount of elements between t_1 and t_x, t_y and t_{i-1}, t_i and t_z, t_w and t_{j-1}, and t_j and t_k, respectively:

$$T = \underbrace{\overbrace{t_1 \dots t_x}^{a} \dots\dots \overbrace{t_y \dots t_{i-1}}^{b} \overbrace{t_i \dots t_z}^{c} \dots\dots \overbrace{t_w \dots t_{j-1}}^{d}}_{\ell} \overbrace{t_j \dots t_k}^{e} \dots\dots t_n. \tag{6.1}$$

With the previous notation, we have that

$$T \cdot \tau_p = \underline{\overbrace{t_i \dots t_z}^{c} \dots\dots \overbrace{t_w \dots t_{j-1}}^{d}} \; \underline{\overbrace{t_1 \dots t_x}^{a} \dots\dots \overbrace{t_y \dots t_{i-1}}^{b}} \overbrace{t_j \dots t_k}^{e} \dots\dots t_n.$$

We say that τ_p caused a *split* if $t_{i-1} = t_i$ or $t_{j-1} = t_j$, and we say that it caused a *merge* if $t_{j-1} = t_1$ or $t_{i-1} = t_j$. Therefore, there can be at most 2 splits and 2 merges when τ_p happens. We will analyse what happens with $\Delta = P(T) - P(T \cdot \tau_p)$ in each combination of splits and merges. For now, we will consider that either there is at least one block between t_x and t_y or $t_x = t_y - 1$ (that is, a and b do not overlap) and that either there is at least one block between t_z and t_w or $t_z = t_w - 1$ (that is, c and d do not overlap). It is easy to see that Δ depends only on the values of a, b, c, d, and e. In the following, all conclusions are obtained considering that $(x+y)^\alpha \leq x^\alpha + y^\alpha$, $x^\alpha - (x+y)^\alpha \leq 0$, $a^\alpha + b^\alpha + c^\alpha + d^\alpha \leq \ell^\alpha$, and $(a+b+c+d)^\alpha \leq \ell^\alpha$:

1. 2 splits and 2 merges: $\Delta = \frac{1}{4}(a^\alpha + (b+c)^\alpha + (d+e)^\alpha - c^\alpha - (d+a)^\alpha - (b+e)^\alpha)$ $\leq \frac{1}{4}(a^\alpha + (b+c)^\alpha + d^\alpha + e^\alpha - c^\alpha - (d+a)^\alpha - (b+e)^\alpha) \leq \frac{1}{4}(a^\alpha + (b+c)^\alpha - c^\alpha)$ $\leq \frac{1}{4}(a^\alpha + b^\alpha) \leq \frac{1}{4}(2\ell^\alpha) < \ell^\alpha$;

2. 2 splits and 1 merge:

(a) $t_{j-1} = t_1$ and $t_{i-1} \neq t_j$: $\Delta = \frac{1}{4}(a^\alpha + (b+c)^\alpha + (d+e)^\alpha - c^\alpha - (d+a)^\alpha - b^\alpha - e^\alpha) \leq \frac{1}{4}((b+c)^\alpha + (d+e)^\alpha - c^\alpha - b^\alpha - e^\alpha) \leq \frac{1}{4}((d+e)^\alpha - e^\alpha) \leq \frac{1}{4}(d^\alpha) < \ell^\alpha$;

(b) $t_{j-1} \neq t_1$ and $t_{i-1} = t_j$: $\Delta = \frac{1}{4}(a^\alpha + (b+c)^\alpha + (d+e)^\alpha - c^\alpha - d^\alpha - a^\alpha - (b+e)^\alpha) \leq \frac{1}{4}(b^\alpha + e^\alpha - (b+e)^\alpha) \leq \frac{1}{4}(b^\alpha) < \ell^\alpha$;

3. 1 split and 2 merges:

(a) $t_{i-1} = t_i$ and $t_{j-1} \neq t_j$: $\Delta = \frac{1}{4}(a^\alpha + (b+c)^\alpha + d^\alpha + e^\alpha - c^\alpha - (d+a)^\alpha - (b+e)^\alpha) \leq \frac{1}{4}(a^\alpha + (b+c)^\alpha - c^\alpha) \leq \frac{1}{4}(a^\alpha + b^\alpha) \leq \frac{1}{4}(2\ell^\alpha) < \ell^\alpha$;

(b) $t_{i-1} \neq t_i$ and $t_{j-1} = t_j$: $\Delta = \frac{1}{4}(a^\alpha + b^\alpha + c^\alpha + (d+e)^\alpha - c^\alpha - (d+a)^\alpha - (b+e)^\alpha) \leq \frac{1}{4}(a^\alpha + b^\alpha + d^\alpha + e^\alpha - (d+a)^\alpha - (b+e)^\alpha) \leq \frac{1}{4}(a^\alpha + b^\alpha) < \ell^\alpha$;

4. 1 split and 1 merge:

(a) $t_{i-1} = t_i$, $t_{j-1} \neq t_j$, $t_{j-1} = t_1$, and $t_{i-1} \neq t_j$: $\Delta = \frac{1}{4}(a^\alpha + (b+c)^\alpha + d^\alpha + e^\alpha - c^\alpha - (d+a)^\alpha - b^\alpha - e^\alpha) \leq \frac{1}{4}((b+c)^\alpha + d^\alpha - c^\alpha - b^\alpha) \leq \frac{1}{4}(d^\alpha) < \ell^\alpha$;

(b) $t_{i-1} = t_i$, $t_{j-1} \neq t_j$, $t_{j-1} \neq t_1$, and $t_{i-1} = t_j$: $\Delta = \frac{1}{4}(a^\alpha + (b+c)^\alpha + d^\alpha + e^\alpha - c^\alpha - d^\alpha - a^\alpha - (b+e)^\alpha) \leq \frac{1}{4}((b+c)^\alpha + e^\alpha - c^\alpha - (b+e)^\alpha) \leq \frac{1}{4}((b+c)^\alpha - c^\alpha) \leq \frac{1}{4}(b^\alpha) < \ell^\alpha$;

(c) $t_{i-1} \neq t_i$, $t_{j-1} = t_j$, $t_{j-1} = t_1$, and $t_{i-1} \neq t_j$: $\Delta = \frac{1}{4}(a^\alpha + b^\alpha + c^\alpha + (d+e)^\alpha - c^\alpha - (d+a)^\alpha - b^\alpha - e^\alpha) \leq \frac{1}{4}((d+e)^\alpha - e^\alpha \leq \frac{1}{4}(d^\alpha) < \ell^\alpha$;

(d) $t_{i-1} \neq t_i$, $t_{j-1} = t_j$, $t_{j-1} \neq t_1$, and $t_{i-1} = t_j$: $\Delta = \frac{1}{4}(a^\alpha + b^\alpha + c^\alpha + (d+e)^\alpha - c^\alpha - d^\alpha - a^\alpha - (b+e)^\alpha) \leq \frac{1}{4}(b^\alpha + (d+e)^\alpha - d^\alpha - (b+e)^\alpha) \leq \frac{1}{4}(b^\alpha + e^\alpha - (b+e)^\alpha) \leq \frac{1}{4}(b^\alpha) < \ell^\alpha$;

5. 0 splits and 2 merges: $\Delta = \frac{1}{4}(a^\alpha + b^\alpha + c^\alpha + d^\alpha + e^\alpha - c^\alpha - (d+a)^\alpha - (b+e)^\alpha) \leq \frac{1}{4}(a^\alpha + b^\alpha) < \ell^\alpha$;

6. 0 splits and 1 merge:

(a) $t_{j-1} = t_1$ and $t_{i-1} \neq t_j$: $\Delta = \frac{1}{4}(a^\alpha + b^\alpha + c^\alpha + d^\alpha + e^\alpha - c^\alpha - (d+a)^\alpha - b^\alpha - e^\alpha) \leq \frac{1}{4}(a^\alpha) < \ell^\alpha$;

(b) $t_{j-1} \neq t_1$ and $t_{i-1} = t_j$: $\Delta = \frac{1}{4}(a^\alpha + b^\alpha + c^\alpha + d^\alpha + e^\alpha - c^\alpha - d^\alpha - a^\alpha - (b+e)^\alpha) \leq \frac{1}{4}(b^\alpha) < \ell^\alpha$.

Cases with 2 or 1 splits and 0 merges clearly evaluate $\Delta \leq 0$, while a 0 split with a 0 merge evaluates $\Delta = 0$. Cases where a and b or c and d overlap are very similar to and easier than the cases above.

We can use this same approach to prove that $c^\alpha_{pr}(T) \geq P(T)$ and $c^\alpha_{prt}(T) \geq P(T)$. □

Breakpoints and strips are extensively used when $\alpha = 0$, as we could see in Chapters 3 and 4. Lemma 46 shows that we can also give lower bounds using the number of breakpoints when $\alpha > 0$.

Lemma 46. *For any unsigned permutation π and $\alpha > 0$,*

$$d^{\alpha}_{pr}(\pi) \geq 2^{\alpha} b_{upr}(\pi),\ d^{\alpha}_{pt}(\pi) \geq 2^{\alpha}\frac{b_p(\pi)}{2},\ \textit{and}\ d^{\alpha}_{prt}(\pi) \geq 2^{\alpha}\frac{b_{upr}(\pi)}{2}.$$

Proof. One prefix reversal can remove at most one upsr-breakpoint. Therefore, to sort a permutation only with prefix reversals, at least $b_{upr}(\pi)$ such rearrangements will be needed. Since each of these reversals will have length $\ell \geq 2$, the cost of each of them should also be $\ell^{\alpha} \geq 2^{\alpha}$. Therefore, to sort π the cost is at least $2^{\alpha} b_{upr}(\pi)$.

A similar analysis can be done for sorting only with prefix transpositions or sorting with both prefix reversals and transpositions. In these cases, since one prefix transposition can remove at most two prefix breakpoints, at least $b_p(\pi)/2$ or $b_{upr}(\pi)/2$ rearrangements will be needed in each of these two problems. □

In the next sections we will present specific algorithms for SbWPR, SbWPSR, SbWPT, SbWPST, SbWPRT, and SbWPSRT, and then show how they can be adapted to the signed problems SbWP$\bar{\text{R}}$, SbWPS$\bar{\text{R}}$, SbWP$\bar{\text{R}}$T, and SbWPS$\bar{\text{R}}$T. In Section 6.1 we show approximation algorithms for unsigned permutations when $\alpha = 1$. In Section 6.2 we present bounds for the diameters of some problems considering $\alpha > 0$. In Sections 6.3 and 6.4 we show that the algorithms of Section 6.1 can also guarantee an approximation factor when $\alpha \neq 1$. We also show in Section 6.3 that an algorithm for the Pancake Flipping problem can be used when $0 < \alpha < 1$ and can guarantee an approximation factor. In Section 6.5 we present some experimental results that we ran over some of the presented algorithms. In Section 6.6 we present some new results for SbWR in order to fulfill the known results for such problem. In Section 6.7 we show results for SbWT and SbWRT. In Section 6.8 we show results for sorting by length-weighted prefix and suffix versions of reversals and transpositions when signed permutations are being considered. Lastly, in Section 6.9 we give a summary of all results presented in this chapter.

6.1 Sorting Algorithms Considering $\alpha = 1$

First, one should note that a sorting sequence for the traditional SbPR is also a valid sorting sequence for SbWPR, although it is not necessarily the optimal sequence. For SbPR, the identity permutation is the only permutation with zero upr-breakpoints, which indicates that to sort a permutation one must reduce its amount of breakpoints. Now notice that this also happens for SbWPR. Therefore, our first idea is to adapt the algorithms for SbPR that already exist.

There is a simple 3-approximation algorithm for SbPR called `3-PR` which, at each step while the permutation is not sorted, finds the highest element that is not in the correct position, brings the strip that contains it to the beginning of the permutation if necessary, inverts the strip if necessary, and puts the strip in the correct place. Such algorithm and its general idea will be used in some of the next sections, so we present it on Algorithm 13.

If we simply use `3-PR` for SbWPR and consider that there are n valid elements, it is easy to see that it performs $O(n)$ main steps (the while loop on Algorithm 13), each one costing at most $3n$ (because there are at most three prefix reversals on each main

Algorithm 13 A 3-approximation algorithm for SBPR.

3-PR(π, n)

Input: permutation π and its size n
Output: cost used to sort π

```
1  c ← 0
2  while π ≠ ι_n do
3      Let i be the highest element such that π_i ≠ i
4      Let b be the position of the first element of the strip that contains i
5      Let e be the position of the last element of the strip that contains i
6      if b ≠ 1 then        // Bring the strip to the beginning if necessary
7          π ← π · ρ_p(e)
8          c ← c + f(e)
9      if π_1 ≠ i then      // Put i in the first position if necessary
10         π ← π · ρ_p(e − b + 1)
11         c ← c + f(e − b + 1)
       // Move i to its right position
12     π ← π · ρ_p(i)
13     c ← c + f(i)
14 return c
```

step), thus guaranteeing that $d^{\alpha}_{pr}(\pi)$ is $O(n^2)$, which, along with the lower bound given in Lemma 43, shows that this is a $O(n)$-approximation. On the other hand, the permutation $\pi^{20}_{*} = (11\ 6\ 12\ 2\ 13\ 7\ 14\ 4\ 15\ 8\ 16\ 1\ 17\ 9\ 18\ 5\ 19\ 10\ 20\ 3)$ is sorted by this algorithm with a total cost of exactly $20^2 - 1$. In fact, Algorithm 14 shows how to build π^n_* for any $n \geq 3$, a family of permutations that are sorted with a cost of $n^2 - 1$ by this algorithm, showing that its approximation factor is also $\Omega(n)$ due to the lower bound.

Algorithm 14 Algorithm to build π^n_*, for $n \geq 3$.

BUILD_PERMUTATION_$\pi^n_*(n)$

Input: integer $n \geq 3$
Output: permutation of the form π^n_*

```
1  π ← (2 1 3 4 5 ... n)
2  for i ← 3 to n do
3      π ← π · ρ_p(i) · ρ_p(i − 1)
4  return π
```

In order to improve the results, we thought about compensating a big prefix reversal that is needed to put an element k in its position by performing smaller prefix reversals that increase the strip that contains k, if these elements show up before k in the permutation. This led to the recursive algorithm `WPRm`, shown in Algorithms 15 and 16.

Example 18. *The following example shows the execution of* `WPRm` *over* $\pi = (5\ 3\ 4\ 7\ 6\ 1\ 2)$. *The highest element that is not in its correct position is* 7 *and its strip also contains the element* 6. *Therefore, the algorithm will try to put the element* 5 *next to them. Recursively, the algorithm will try to increase the strip that contains* 5 *with elements that come before it, which is not possible since it is already in the beginning. Since there is no need to invert its strip, it will just put* 5 *next to* 7, *generating* $\pi = (4\ 3\ 5\ 7\ 6\ 1\ 2)$. *The recursive call for*

5 *returns, and now the algorithm must bring the strip with* 7 *to the beginning, invert it, and then move it to position* 7*, generating* $\pi = (2\ 1\ 4\ 3\ 5\ 6\ 7)$*. Now, the highest element that is not in the correct position is* 4*, whose strip contains* 3*. In order to put* 2 *next to this strip, it suffices to invert the strip that contains* 2*, so the next permutation is* $\pi = (1\ 2\ 4\ 3\ 5\ 6\ 7)$*. After the recursive call, the algorithm only has to put the strip that contains* 4 *in the right position by bringing it to the beginning, inverting it, and finally putting it in position* 4*, generating the identity. Note that the sequence used by* `WPRm` *to sort this permutation has a cost of 29.*

Algorithm 15 Sorting algorithm for SbWPR.

WPRm(π, n)

Input: permutation π and its size n
Output: cost used to sort π

```
1  c ← 0
2  while π ≠ ι_n do
3      Let k be the highest element such that π_k ≠ k
4      c ← c + recWPRm(π, n, k, k) // cost of placing k at position k
5  return c
```

Algorithm 16 Auxiliary algorithm for `WPRm`.

recWPRm(π, n, k, p)

Input: permutation π, its size n, element k, and position p
Output: cost to put k in position p

```
 1  if p < 1 or k < 1 then      // Base case: values of position and element only
                                // decrease, so stop if any of them becomes zero
 2      return 0
 3  Let ini and end be the first and last positions of the strip that contains k
 4  if end > p then             // Base case: we only try to increase the strip if element
                                // k appears before position p, where it should be put at
 5      return 0
 6  k' ← min{π_ini, π_end} − 1
 7  c ← recWPRm(π, n, k', ini − 1) // Increase the strip that contains k
 8  if ini ≠ 1 then             // Bring the strip with k to the beginning if necessary
 9      π ← π · ρ_p(end)
10      c ← c + f(end)
11  if π_{end−ini+1} > π_1 then // Revert strip if necessary
12      π ← π · ρ_p(end − ini + 1)
13      c ← c + f(end − ini + 1)
14  if p > 1 then               // Move strip with k to position p
15      π ← π · ρ_p(p)
16      c ← c + f(p)
17  return c
```

Consider now the permutation $\pi_\star^{20} = (15\ 18\ 11\ 14\ 7\ 10\ 3\ 6\ 2\ 4\ 1\ 8\ 5\ 12\ 9\ 16\ 13\ 20\ 17\ 19)$. Note that `WPRm` cannot increase the strip that contains 20 because the element 19 appears after it. Therefore, it will just bring 20 to the beginning of the permutation and put it in its correct position. This will put 19 in position 1 of the permutation,

so that in the next step the algorithm will just put it in its correct position. Now the permutation is (13 16 9 12 5 8 1 4 2 6 3 10 7 14 11 18 15 17 19 20) and something similar happens: it is not possible to increase the strip that contains 18, so `WPRm` brings it to the beginning and puts it in position 18; so, 17 is in the first position and it is just placed in its correct position. It is not hard to see that this will happen for all other values of i, for $i \in \{16, 14, 12, \ldots, 6, 4\}$. After this, the permutation will be $(2\ 1\ 3\ 4\ 5\ \ldots\ n)$ and one extra prefix reversal is needed to finish the sorting. The total cost used by `WPRm` to sort $\pi_\star^{20}$ was of 299. Now note that it is possible to find permutations $\pi_\star^n$ for any even value of $n \geq 6$ for which the cost that `WPRm` will use to sort it is exactly $3n^2/4 - 1$ (Algorithm 17 shows how to build $\pi_\star^n$). This family of permutations shows that the approximation factor of this algorithm, considering the lower bound of n, is also in $\Omega(n)$ (more precisely, it cannot be better than $3n/4 - 1/n$).

Algorithm 17 Algorithm to build $\pi_\star^n$, for $n \geq 6$ and even.

```
BUILD_PERMUTATION_π*_n(n)
    Input: an even integer n ≥ 6
    Output: permutation of the form π*_n
 1  π_{n-2} ← n; π_n ← n − 1; π_2 ← n − 2; π_{n-1} ← n − 3
 2  e ← 1; d ← n − 4; i ← n − 4
 3  while i > 4 do
 4      π_d ← i; π_e ← i − 1; π_{e+3} ← i − 2; π_{d+1} ← i − 3
 5      d ← d − 2; e ← e + 2; i ← i − 4
 6  if n ≡ 0 (mod 4) then
 7      π_d ← 4; π_e ← 3; π_{e+2} ← 2; π_{d+1} ← 1
 8  else
 9      π_e ← 1; π_{d+1} ← 2
10  return π
```

While the permutation is not sorted, the algorithm `2-PR` for SBPR [30], mentioned in Section 4.1, first tries to apply a 1-move and, if that is not possible, then it tries to apply a 0-move followed by a 1-move. If none of these is possible, then the permutation has a special format, which can be sorted with twice its number of breakpoints. The algorithm we adapt considering this one is called `WPRg`: it first searches for all sequences of one or two prefix reversals that remove one breakpoint and, if there is at least one sequence, it chooses the one with minimum cost; otherwise, the special format is reached and it uses the same sequence of rearrangements described by Fischer and Ginzinger [30].

Example 19. *The following example shows the execution of* `WPRg` *over* $\pi = (5\ 3\ 4\ 7\ 6\ 1\ 2)$ *of the previous example. No 1-move can be applied, so the sequence* $\rho_p(3)$, $\rho_p(2)$ *is chosen because it has the minimum cost among the other sequences of two prefix reversals that remove one upr-breakpoint. So, we have* $(3\ 4\ 5\ 7\ 6\ 1\ 2)$ *and the sequence* $\rho_p(5)$, $\rho_p(2)$ *is chosen, generating* $(7\ 6\ 5\ 4\ 3\ 1\ 2)$. *At last, sequence* $\rho_p(7)$, $\rho_p(2)$ *is chosen, we finish the sorting, and a total cost of 21 was used by the algorithm.*

Note that `WPRg` also uses $O(n)$ main steps, each one with a cost of at most $2n$. Therefore, $\texttt{WPRg}(\pi, n) \leq 2n\, O(n) = O(n^2)$. Considering the lower bound of n, `WPRg`'s approximation factor is $O(n^2)/n$, which is $O(n)$.

Based on the algorithms for the traditional problems SbPT, SbPRT, SbPSR, SbPST, SbPSRT, SbP$\bar{\text{R}}$, SbP$\bar{\text{R}}$T SbPS$\bar{\text{R}}$, and SbPS$\bar{\text{R}}$T, and based on the idea described for `WPRg`, we can developed $O(n)$-approximation algorithms for all the other problems we are considering (called `WPTg`, `WPRTg`, `WPSRg`, `WPSTg`, `WPSRTg`, `WP$\bar{\text{R}}$g`, `WP$\bar{\text{R}}$Tg`, `WPS$\bar{\text{R}}$g`, and `WPS$\bar{\text{R}}$Tg`, respectively). These algorithms are greedy in the sense of removal of breakpoints and of cost of the rearrangement(s) performed, and that is why their names end with a "g".

For the rest of this section, we describe algorithms which are not adaptations of existing algorithms. We used a divide-and-conquer strategy in order to improve the approximation factors to $O(\lg^2 n)$.

6.1.1 Sorting by Length-Weighted Prefix Reversals

In a similar way regarding the algorithms presented by Pinter and Skiena [50] and by Bender *et al.* [5], the main idea of our algorithm is to partition the permutation in two halves and to use the algorithm itself to sort each part. Unlike them, we can only do this by using prefix reversals and, since a reversal can be mimicked by at most three prefix reversals, a simple adaptation of their algorithms is not viable for this specific problem, as the following example shows. Consider again $\pi = (5\ 3\ 4\ 7\ 6\ 1\ 2)$. One of the reversals performed by the algorithm of Pinter and Skiena [50] is $\rho(5, 7)$. This specific reversal is mimicked by the three prefix reversals $\rho_p(7)$, $\rho_p(3)$, and $\rho_p(7)$. Therefore, what before had a cost of 3, now has a cost of 17, which notably is more than three times worse.

Given a permutation π with n valid elements, we can transform it into the identity permutation by moving the elements that are greater than the median $\lceil \frac{n}{2} \rceil$ to the beginning of the permutation, moving the elements that are less than or equal to the median to the end of the permutation, and then we can sort the first half in decreasing order, reverse the whole permutation, and sort the second half (now in the beginning) in increasing order.

Therefore, we have that such an algorithm should be able to do two types of sorting: one that leaves an interval of the permutation in increasing order (INC) and one that leaves an interval in decreasing order (DEC). From the explanation above, we can see that whenever a sorting in increasing order needs to be done, we want to partition the permutation so that the elements that are greater than the median stay at the beginning of the interval and the ones that are less than or equal to the median stay at the end. We will say that this kind of partition is of type DEC, because regarding the median the values will be decreasing. Equivalently, when a decreasing sorting needs to be done, we want the elements that are less than or equal to the median at the beginning and the ones that are greater than the median at the end. We will say that this partition is of type INC, again because regarding the median the values will be increasing. Figure 6.1 reinforces the idea of how an interval from 1 to some n' of a permutation must be returned by the partition algorithm to the sorting algorithm.

Since both partition and sorting algorithms are recursive, they must act on an interval of the permutation (which, of course, always starts at 1). We will say that n is the size of the permutation (valid elements) and n' is the final position of the interval where the algorithms must act. Algorithm 18 presents our sorting algorithm, `WPR`. We are considering that INC and DEC are represented by constants 0 and 1, respectively. Note that the first

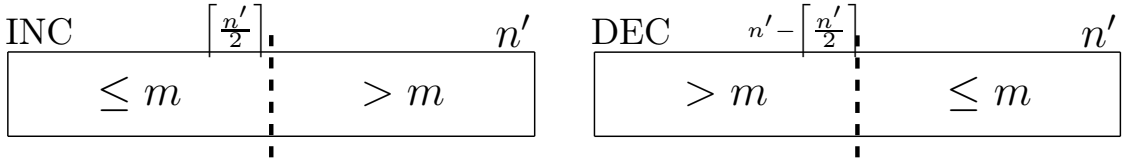


Figure 6.1: Format of the interval $[1..n']$ of a permutation after the partition algorithm is called by the sorting algorithm. The left one is the result of an INC partition, which is needed by a DEC sorting. The right one is the result of a DEC partition, which is needed by an INC sorting. The median of the elements contained in π_1 through $\pi_{n'}$ is m.

call to our algorithm must be WPR$(\pi, n,$ INC) if one wants to transform π into the identity permutation.

Algorithm 18 An $O(\lg^2 n)$-approximation algorithm for SbWPR.

WPR(π, n', *type*)

Input: permutation π, integer n', and *type* of the sorting (INC=0 or DEC=1)
Output: cost to sort the interval from 1 to n' of π according to the *type* required

1 **if** $\pi_1, \pi_2, \ldots, \pi_{n'}$ is sorted in *type* form **then**
// Base case: interval from 1 to n' is already sorted according to *type*
2 **return** 0
3 **if** $n' \leq 1$ **then** // Base case: permutation with one or no element
4 **return** 0
5 **if** $n' = 2$ **then** // Base case: permutation with two elements
6 **if** ($\pi_1 > \pi_2$ and *type* = INC) or ($\pi_1 < \pi_2$ and *type* = DEC) **then**
7 $\pi \leftarrow \pi \cdot \rho_p(2)$
8 **return** $f(2)$
9 **return** 0
10 $m \leftarrow \min\{\pi_1, \pi_2, \ldots, \pi_{n'}\} - 1 + \lceil n'/2 \rceil$
11 $c \leftarrow$ PARTITIONWPR$(\pi, n', 1 - type, m)$ // Partition π using median m as the pivot
12 **if** *type* = INC **then** // Calculating size d of the first part
13 $d \leftarrow n' - \lceil n'/2 \rceil$
14 **else**
15 $d \leftarrow \lceil n'/2 \rceil$
16 $c \leftarrow c +$ WPR$(\pi, d, 1 - type)$ // Sort from 1 to d with opposite type
17 $\pi \leftarrow \pi \cdot \rho_p(n')$
18 $c \leftarrow c + f(n')$
19 $c \leftarrow c +$ WPR$(\pi, n' - d, type)$ // Sort from 1 to $n' - d$ with the same type
20 **return** c

Example 20. *The following example shows the execution of* WPR *over* $\pi =$ (9 6 2 14 10 15 7 12 4 11 3 8 13 1 5) *(the detail of the first partition call will be given in the next*

example). We will say that the first call, which is WPR(π,15,INC) *is call (0).*

π = (9 6 2 14 10 15 7 12 4 11 3 8 13 1 5) // *no base case applies, so calls partition*
$\pi \leftarrow$partitionWPR(π,15,DEC,8) = (15 10 9 14 12 13 11 4 3 8 7 2 6 1 5) // *(1) calls* WPR(π,7,DEC)*; no base case*
$\pi \leftarrow$partitionWPR(π,7,INC,12) = (10 9 12 11 13 15 14 4 3 8 7 2 6 1 5) // *(2) calls* WPR(π,4,INC)*; no base case*
$\pi \leftarrow$partitionWPR(π,4,DEC,10) = (11 12 9 10 13 15 14 4 3 8 7 2 6 1 5) // *(3) calls* WPR(π,2,DEC)*; has base case*
$\pi \leftarrow \pi \cdot \rho_p(2)$ = (12 11 9 10 13 15 14 4 3 8 7 2 6 1 5) // *returns to (2); reverse whole interval*
$\pi \leftarrow \pi \cdot \rho_p(4)$ = (10 9 11 12 13 15 14 4 3 8 7 2 6 1 5) // *(4) calls* WPR(π,4,DEC)*; has base case*
$\pi \leftarrow \pi \cdot \rho_p(2)$ = (9 10 11 12 13 15 14 4 3 8 7 2 6 1 5) // *returns to (1); reverse whole interval*
$\pi \leftarrow \pi \cdot \rho_p(7)$ = (14 15 13 12 11 10 9 4 3 8 7 2 6 1 5) // *(5) calls* WPR(π,3,DEC)*; no base case*
$\pi \leftarrow$partitionWPR(π,3,INC,14) = (13 14 15 12 11 10 9 4 3 8 7 2 6 1 5) // *(6) calls* WPR(π,2,INC)*; is* INC *sorted*
π = (13 14 15 12 11 10 9 4 3 8 7 2 6 1 5) // *returns to (5); reverse whole interval*
$\pi \leftarrow \pi \cdot \rho_p(3)$ = (15 14 13 12 11 10 9 4 3 8 7 2 6 1 5) // *(7) calls* WPR(π,1,DEC)*; is* DEC *sorted*
π = (15 14 13 12 11 10 9 4 3 8 7 2 6 1 5) // *returns to (0); reverse whole interval*
$\pi \leftarrow \pi \cdot \rho_p(15)$ = (5 1 6 2 7 8 3 4 9 10 11 12 13 14 15) // *(8) calls* WPR(π,8,INC)*; no base case*
$\pi \leftarrow$partitionWPR(π,8,DEC,4) = (5 6 8 7 2 1 3 4 9 10 11 12 13 14 15) // *(9) calls* WPR(π,4,DEC)*; no base case*
$\pi \leftarrow$partitionWPR(π,4,INC,6) = (5 6 8 7 2 1 3 4 9 10 11 12 13 14 15) // *(10) calls* WPR(π,2,INC)*; is* INC *sorted*
π = (5 6 8 7 2 1 3 4 9 10 11 12 13 14 15) // *returns to (9); reverse whole interval*
$\pi \leftarrow \pi \cdot \rho_p(4)$ = (7 8 6 5 2 1 3 4 9 10 11 12 13 14 15) // *(11) calls* WPR(π,2,DEC)*; has base case*
$\pi \leftarrow \pi \cdot \rho_p(2)$ = (8 7 6 5 2 1 3 4 9 10 11 12 13 14 15) // *returns to (8); reverse whole interval*
$\pi \leftarrow \pi \cdot \rho_p(8)$ = (4 3 1 2 5 6 7 8 9 10 11 12 13 14 15) // *(12) calls* WPR(π,4,INC)*; no base case*
$\pi \leftarrow$partitionWPR(π,4,DEC,2) = (4 3 1 2 5 6 7 8 9 10 11 12 13 14 15) // *(13) calls* WPR(π,2,DEC)*; is* DEC *sorted*
π = (4 3 1 2 5 6 7 8 9 10 11 12 13 14 15) // *returns to (12); reverse whole interval*
$\pi \leftarrow \pi \cdot \rho_p(4)$ = (2 1 3 4 5 6 7 8 9 10 11 12 13 14 15) // *(14) calls* WPR(π,2,INC)*; has base case*
$\pi \leftarrow \pi \cdot \rho_p(2)$ = (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)

Figure 6.2 shows the main features of the partition algorithm, explained next. The idea is (1) to partition the first half of the given interval with the opposite type considering the type that was given (that is, if one asks for an INC partition, we partition the first half with the type DEC), (2) to reverse the entire interval, (3) to partition the other half also with the opposite type, and (4) if m is the pivot of the partition, after the recursive calls one must find positions j and k for which $\pi_\ell \leq m$ (resp. $\pi_\ell > m$) for all $j \leq \ell \leq k$, if the partition type is INC (resp. DEC), and apply a rearrangement $\rho_p(k)$, if $j > 1$, to finish the partition of the interval from 1 to n'. We point out that the partition algorithm considers the set of elements E_P of a given interval and it must partition it regarding a pivot m that is not necessarily the median of E_P or even is in E_P. However, such pivot is the median of the set of elements E_S of the interval that is being considered by the sorting algorithm.

The first base case of the partition is trivial: we do nothing if $n' \leq 1$. In order to improve the performance, we added a second case that tries to discover whether the permutation is almost partitioned, as shown in Figure 6.3 and described next. If the partition is of type INC (resp. DEC), (1) we find a position $x \geq 0$ for which $\pi_\ell \leq m$ (resp. $\pi_\ell > m$) for all $1 \leq \ell \leq x$ and a position $y \leq n' + 1$ for which $\pi_\ell > m$ (resp. $\pi_\ell \leq m$) for all $y \leq \ell \leq n'$. Note that x and y always exist, because one can always have $x = 0$ and/or $y = n' + 1$. So, (2) we try to find a position z for which $\pi_\ell > m$ (resp. $\pi_\ell \leq m$) for all $x < \ell \leq z$ and $\pi_\ell \leq m$ (resp. $\pi_\ell < m$) for all $z < \ell < y$. If position z exists, then rearrangements (3) $\rho_p(z)$ (if $x > 0$) and (4) $\rho_p(y - 1)$ are sufficient to finish the partition. Otherwise, we can at least update n' to $y - 1$. Algorithm 19 shows the partition algorithm, `partitionWPR`, described above.

Example 21. *The following example shows the execution of* `partitionWPR` *over* π = (9 6 2 14 10 15 7 12 4 11 3 8 13 1 5)*. We are considering the previous example, so the first call (call (0)) is* partitionWPR(π, 15, *DEC,* 8)*. Also, note that if no base case occurs,*

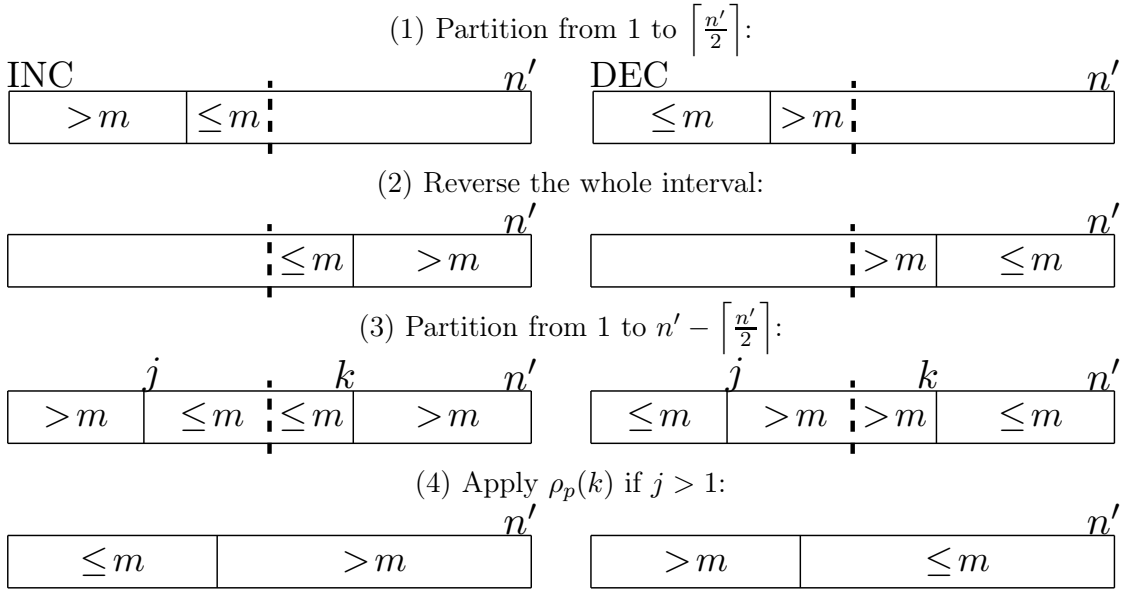


Figure 6.2: Main idea of the partition algorithm for SbWPR, regarding a pivot m, over the interval from position 1 to n' of a permutation. The left column represents the partition of type INC while the right column represents the partition of type DEC.

then the end of the interval may be updated:

π = (9 6 2 14 10 15 7 12 4 11 3 8 13 1 5) // *(1) calls* `partitionWPR`*(π,7,INC,8); no base case*
π = (9 6 2 14 10 15 7 12 4 11 3 8 13 1 5) // *(2) calls* `partitionWPR`*(π,4,DEC,8); has base case*
$\pi \leftarrow \pi{\cdot}\rho_p(3)$ = (2 6 9 14 10 15 7 12 4 11 3 8 13 1 5) // *still treating base case*
$\pi \leftarrow \pi{\cdot}\rho_p(4)$ = (14 9 6 2 10 15 7 12 4 11 3 8 13 1 5) // *returns to (1); reverses whole interval*
$\pi \leftarrow \pi{\cdot}\rho_p(7)$ = (7 15 10 2 6 9 14 12 4 11 3 8 13 1 5) // *(3) calls* `partitionWPR`*(π,3,DEC,8); has base case*
$\pi \leftarrow \pi{\cdot}\rho_p(3)$ = (10 15 7 2 6 9 14 12 4 11 3 8 13 1 5) // *returns to (1); finishes partition until position 7*
$\pi \leftarrow \pi{\cdot}\rho_p(5)$ = (6 2 7 15 10 9 14 12 4 11 3 8 13 1 5) // *returns to (0); reverses whole interval*
$\pi \leftarrow \pi{\cdot}\rho_p(13)$ = (13 8 3 11 4 12 14 9 10 15 7 2 6 1 5) // *(4) calls* `partitionWPR`*(π,6,INC,8); no base case*
π = (13 8 3 11 4 12 14 9 10 15 7 2 6 1 5) // *(5) calls* `partitionWPR`*(π,3,DEC,8); is DEC partitioned*
π = (13 8 3 11 4 12 14 9 10 15 7 2 6 1 5) // *returns to (4); reverses whole interval*
$\pi \leftarrow \pi{\cdot}\rho_p(5)$ = (4 11 3 8 13 12 14 9 10 15 7 2 6 1 5) // *(6) calls* `partitionWPR`*(π,2,DEC,8); has base case*
$\pi \leftarrow \pi{\cdot}\rho_p(2)$ = (11 4 3 8 13 12 14 9 10 15 7 2 6 1 5) // *returns to (4); finishes partition until position 6*
$\pi \leftarrow \pi{\cdot}\rho_p(4)$ = (8 3 4 11 13 12 14 9 10 15 7 2 6 1 5) // *returns to (0); finishes partition until position 15*
$\pi \leftarrow \pi{\cdot}\rho_p(10)$ = (15 10 9 14 12 13 11 4 3 8 7 2 6 1 5)

Lemma 47 analyses the cost used by `WPR` to sort any permutation and Theorem 7 shows that the approximation fator of `WPR` is $O(\lg^2 n)$.

Lemma 47. *For any unsigned permutation π with n valid elements and for $\alpha = 1$, `partitionWPR`(π, n) is $O(n \lg n)$ and `WPR`(π, n) is $O(n \lg^2 n)$.*

Proof. It is easy to see that $\texttt{WPR}(\pi, n) \leq 2 \times \texttt{WPR}(\pi, n/2) + \texttt{partitionWPR}(\pi, n) + O(n)$. Furthermore, $\texttt{partitionWPR}(\pi, n) \leq 2 \times \texttt{partitionWPR}(\pi, n/2) + O(n)$. Therefore, we have that $\texttt{partitionWPR}(\pi, n) = O(n \lg n)$, which means that $\texttt{WPR}(\pi, n) = O(n \lg^2 n)$. □

Theorem 7. *For $\alpha = 1$, SbWPR is $O(\lg^2 n)$-approximable.*

Proof. Regarding the time complexity, first note that a prefix reversal takes time $O(n)$ to be done. In Algorithm 19, which shows the partition algorithm, lines 4, 7, and 19 are all $O(n)$. All other steps from the partition algorithm or the sorting algorithm (except for the recursive calls) are either $O(1)$ or prefix reversals. Therefore, if $T'(n)$ is the time used by `WPR` to sort π and $T''(n)$ is the time used by `partitionWPR` to partition π, then

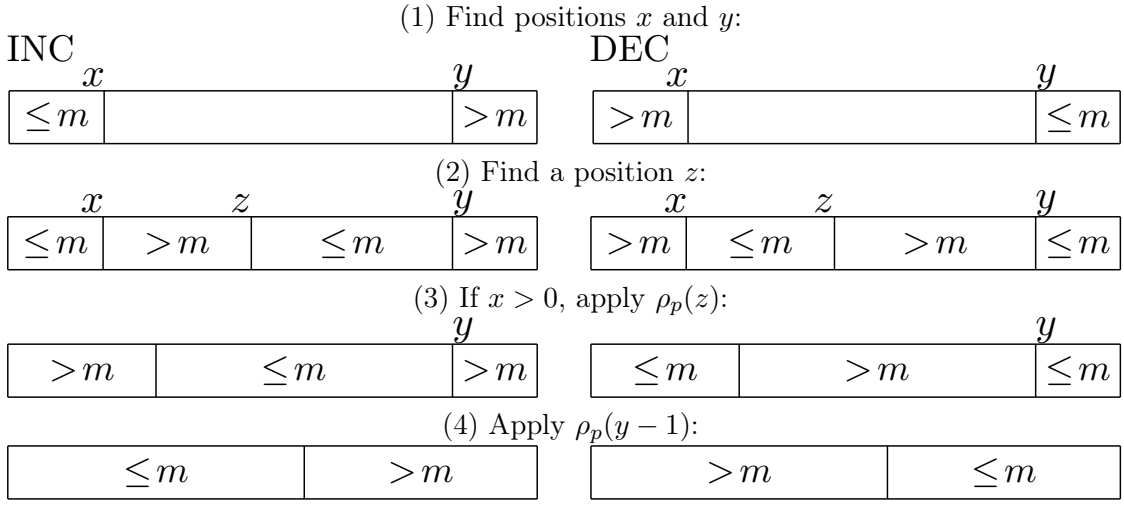


Figure 6.3: Base case of `partitionWPR` with pivot m. The left column represents the partition of type INC while the right column represents the partition of type DEC.

Algorithm 19 Partition algorithm for `WPR`.

PARTITIONWPR(π, n', *type*, m)

Input: permutation π, integer n', *type* (INC/DEC) of the partition, and integer m
Output: cost to partition interval from 1 to n' of π according to *type* and pivot m

1 **if** $n' \leq 1$ **then** // Base case: permutation with one or no element
2 **return** 0
3 $c \leftarrow 0$
4 Let $x \geq 0$ be the highest integer such that, for all $i \in [1..x]$, $\pi_i \leq m$ if *type* = INC (resp. $\pi_i > m$ if *type* = DEC) and let $y \leq n'$ be the smallest integer such that, for all $i \in [y..n']$, $\pi_i > m$ if *type* = INC (resp. $\pi_i \leq m$ if *type* = DEC)
5 **if** $x = y - 1$ **then** // Base case: permutation is already partitioned
6 **return** 0
7 **if** there is an integer z such that $\pi_i > m$ and $\pi_j \leq m$ if *type* = INC (resp. $\pi_i \leq m$ and $\pi_j > m$ if *type* = DEC) for all $i \in [x+1..z]$ and $j \in [z+1..y-1]$ **then**
// Base case: permutation is partitioned in at most four parts
8 **if** $x > 0$ **then**
9 $\pi \leftarrow \pi \cdot \rho_p(z)$
10 $c \leftarrow c + f(z)$
11 $\pi \leftarrow \pi \cdot \rho_p(y-1)$
12 $c \leftarrow c + f(y-1)$
13 **return** c
14 $n' \leftarrow y - 1$
15 $c \leftarrow c +$ PARTITIONWPR($\pi, \left\lceil \frac{n'}{2} \right\rceil, 1 - type, m$) // Partition from 1 to $\left\lceil \frac{n'}{2} \right\rceil$ considering m
16 $\pi \leftarrow \pi \cdot \rho_p(n')$
17 $c \leftarrow c + f(n')$
18 $c \leftarrow c +$ PARTITIONWPR($\pi, n' - \left\lceil \frac{n'}{2} \right\rceil, 1 - type, m$) // Partition from 1 to $n' - \left\lceil \frac{n'}{2} \right\rceil$
19 Let j and k be the first and last positions, respectively, of the interval that contains only elements that are less than or equal to m if *type* = INC or greater than m if *type* = DEC
20 **if** $j > 1$ and $j \leq k$ **then**
21 $\pi \leftarrow \pi \cdot \rho_p(k)$
22 $c \leftarrow c + f(k)$
23 **return** c

$T'(n) \leq 2 \times T'(n/2) + T''(n) + O(n)$ and $T''(n) \leq 2 \times T''(n/2) + O(n)$. Therefore, $T''(n) = O(n \log n)$ and $T'(n) = O(n \log^2 n)$ and the time complexity of `WPR` is $O(n \lg^2 n)$.

Using Lemmas 43 and 47, the approximation factor is $O(n \lg^2 n)/n$, which is $O(\lg^2 n)$. Because n is $O(\bar{n})$, we can also say that SbWPR is $O(\lg^2 \bar{n})$-approximable. □

Note that the partition algorithm can be adapted to sort binary strings, since it separates the permutation in two parts. Furthermore, it guarantees an approximation factor of $O(\lg n)$ for this problem, as Theorem 8 shows.

Theorem 8. *For $\alpha = 1$,* SbWPR *is $O(\lg n)$-approximable.*

Proof. As mentioned in Lemma 47 and Theorem 7, the cost used by `partitionWPR` is $O(n \lg n)$ and it runs in $O(n \lg n)$ time. Using Lemma 43, the lower bound to sort any binary string is n. Therefore, the approximation factor of such algorithm is $O(n \lg n)/n$, which is $O(\lg n)$. □

6.1.2 Sorting by Length-Weighted Prefix and Suffix Reversals

It is not difficult to adapt algorithms `WPR` and `partitionWPR` into `WSR` and `partitionWSR`, which use only suffix reversals. The important difference is that, instead of receiving the end of the interval, now they receive its beginning. With these four algorithms we can make an algorithm called `WPSR` for SbWPSR: if the permutation is separated by positions i and j, as stated in Definition 22, then one can sort from 1 to i in increasing order with `WPR` and sort from j to n in increasing order with `WSR`; if the permutation is not separated, then one can explicitly partition the permutation and let the elements less than or equal to the median at the beginning of the permutation and the ones greater than the median at the end, sort the first half in increasing order with `WPR`, and sort the second half in increasing order with `WSR`.

As Example 17 showed, in the case where the number of valid elements is $n = \bar{n}$ and the permutation is separated, the separation can happen for more than one pair of positions i and j. For implementation purposes we decided to choose the pair for which the distance from i to the median $\lceil \bar{n}/2 \rceil$ is the smallest.

The partition algorithm for this problem, called `partitionWPSR`, works as follows. If no base case happens (the permutation is already partitioned or is partitioned in at most four parts), then we can partition the first half of the permutation in an INC form with `partitionWPR` and considering $\lceil n/2 \rceil$ as the pivot, and then partition the second half with `partitionWSR` in a DEC form also considering $\lceil n/2 \rceil$ as the pivot. This will leave the permutation with at most three parts (the first and the last ones with elements less than or equal to $\lceil n/2 \rceil$ and the second one with elements greater than $\lceil n/2 \rceil$), and an extra suffix reversal can finish the partition.

Example 22. *The following example shows the execution of* `WPSR` *over* $\pi = (9\ 6\ 2\ 14\ 10$

15 7 12 4 11 3 8 13 1 5). *We refer to the first call,* `WPSR`*(*π,15,INC*), as call (0).*

π = (9 6 2 14 10 15 7 12 4 11 3 8 13 1 5) // *no base case and not separated*
π = (9 6 2 14 10 15 7 12 4 11 3 8 13 1 5) // *(1) calls* `partitionWPSR`*(*π,15*); no base case*
π = (9 6 2 14 10 15 7 12 4 11 3 8 13 1 5) // *(2) calls* `partitionWPR`*(*π,8,*INC*,8)
$\pi \leftarrow \pi{\cdot}\rho_p(3)$ = (2 6 9 14 10 15 7 12 4 11 3 8 13 1 5)
$\pi \leftarrow \pi{\cdot}\rho_p(4)$ = (14 9 6 2 10 15 7 12 4 11 3 8 13 1 5)
$\pi \leftarrow \pi{\cdot}\rho_p(7)$ = (7 15 10 2 6 9 14 12 4 11 3 8 13 1 5)
$\pi \leftarrow \pi{\cdot}\rho_p(3)$ = (10 15 7 2 6 9 14 12 4 11 3 8 13 1 5)
$\pi \leftarrow \pi{\cdot}\rho_p(5)$ = (6 2 7 15 10 9 14 12 4 11 3 8 13 1 5) // *returns to (1); (3) calls* `partitionWSR`*(*π,9,DEC,8*)*
$\pi \leftarrow \pi{\cdot}\rho_s(13)$ = (6 2 7 15 10 9 14 12 4 11 3 8 5 1 13)
$\pi \leftarrow \pi{\cdot}\rho_s(9)$ = (6 2 7 15 10 9 14 12 13 1 5 8 3 11 4)
$\pi \leftarrow \pi{\cdot}\rho_s(14)$ = (6 2 7 15 10 9 14 12 13 1 5 8 3 4 11)
$\pi \leftarrow \pi{\cdot}\rho_s(10)$ = (6 2 7 15 10 9 14 12 13 11 4 3 8 5 1) // *returns to (1); finishes the partition*
$\pi \leftarrow \pi{\cdot}\rho_s(4)$ = (6 2 7 1 5 8 3 4 11 13 12 14 9 10 15) // *returns to (0); (4) calls* `WPR`*(*π,8,INC*)*
$\pi \leftarrow \pi{\cdot}\rho_p(2)$ = (2 6 7 1 5 8 3 4 11 13 12 14 9 10 15)
$\pi \leftarrow \pi{\cdot}\rho_p(6)$ = (8 5 1 7 6 2 3 4 11 13 12 14 9 10 15)
$\pi \leftarrow \pi{\cdot}\rho_p(3)$ = (1 5 8 7 6 2 3 4 11 13 12 14 9 10 15)
$\pi \leftarrow \pi{\cdot}\rho_p(5)$ = (6 7 8 5 1 2 3 4 11 13 12 14 9 10 15)
$\pi \leftarrow \pi{\cdot}\rho_p(3)$ = (8 7 6 5 1 2 3 4 11 13 12 14 9 10 15)
$\pi \leftarrow \pi{\cdot}\rho_p(4)$ = (5 6 7 8 1 2 3 4 11 13 12 14 9 10 15)
$\pi \leftarrow \pi{\cdot}\rho_p(4)$ = (8 7 6 5 1 2 3 4 11 13 12 14 9 10 15)
$\pi \leftarrow \pi{\cdot}\rho_p(9)$ = (4 3 2 1 5 6 7 8 11 13 12 14 9 10 15)
$\pi \leftarrow \pi{\cdot}\rho_p(4)$ = (1 2 3 4 5 6 7 8 11 13 12 14 9 10 15) // *returns to (0); (5) calls* `WSR`*(*π,9,INC*)*
$\pi \leftarrow \pi{\cdot}\rho_s(9)$ = (1 2 3 4 5 6 7 8 15 10 9 14 12 13 11)
$\pi \leftarrow \pi{\cdot}\rho_s(12)$ = (1 2 3 4 5 6 7 8 15 10 9 11 13 12 14)
$\pi \leftarrow \pi{\cdot}\rho_s(14)$ = (1 2 3 4 5 6 7 8 15 10 9 11 13 14 12)
$\pi \leftarrow \pi{\cdot}\rho_s(13)$ = (1 2 3 4 5 6 7 8 15 10 9 11 12 14 13)
$\pi \leftarrow \pi{\cdot}\rho_s(10)$ = (1 2 3 4 5 6 7 8 15 13 14 12 11 9 10)
$\pi \leftarrow \pi{\cdot}\rho_s(12)$ = (1 2 3 4 5 6 7 8 15 13 14 10 9 11 12)
$\pi \leftarrow \pi{\cdot}\rho_s(12)$ = (1 2 3 4 5 6 7 8 15 13 14 12 11 9 10)
$\pi \leftarrow \pi{\cdot}\rho_s(14)$ = (1 2 3 4 5 6 7 8 15 13 14 12 11 10 9)
$\pi \leftarrow \pi{\cdot}\rho_s(9)$ = (1 2 3 4 5 6 7 8 9 10 11 12 14 13 15)
$\pi \leftarrow \pi{\cdot}\rho_s(13)$ = (1 2 3 4 5 6 7 8 9 10 11 12 15 13 14)
$\pi \leftarrow \pi{\cdot}\rho_s(14)$ = (1 2 3 4 5 6 7 8 9 10 11 12 15 14 13)
$\pi \leftarrow \pi{\cdot}\rho_s(13)$ = (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)

Algorithms 20 and 21 show `WPSR` and `partitionWPSR`. Theorem 9 shows that `WPSR` is a $O(\lg^2 n)$-approximation.

Algorithm 20 An $O(\lg^2 n)$-approximation algorithm for SbWPSR.

WPSR(π, n)

Input: permutation π and its size n
Output: cost used to sort π

```
1  if n ≤ 1 then
2      return 0
3  if n = 2 then
4      if π_1 > π_2 then
5          π ← π · ρ_p(2)
6          return f(2)
7      return 0
8  if π is separated by i and j then// According to Definition 22
9      c ← c + WPR(π, i, INC) // Sort from 1 to i in increasing order
10     c ← c + WSR(π, j, INC) // Sort from j to n in increasing order
11 else
12     c ← partitionWPSR(π, n)
13     c ← c + WPR(π, ⌈n/2⌉, INC) // Sort from 1 to ⌈n/2⌉ in increasing order
14     c ← c + WSR(π, ⌈n/2⌉ + 1, INC) // Sort from ⌈n/2⌉ + 1 to n in increasing order
15 return c
```

Theorem 9. *For* $\alpha = 1$, SbWPSR *is* $O(\lg^2 n)$*-approximable.*

Proof. Let π be a permutation with $\bar{n}$ elements and n valid elements. If π is separated by positions i and j ($n = i + \bar{n} - j + 1$), according to Definition 22, then `WPSR` depends

on WPR over the segment from 1 to i, and on WSR over the segment from j to $\bar{n}$. Since WSR is equivalent to WPR, these two use a cost of at most $O(n \lg^2 n)$ each to sort π.

If π is not separated, then $n = \bar{n}$ and WPSR depends on partitionWPSR over the whole permutation, on WPR over the first half, and on WSR over the second half. Again, WSR and WPR have a cost of $O(n \lg^2 n)$ to sort π. The partition algorithm depends on the cost of partitionWPR over half the permutation, on the cost partitionWSR over the other half, and a suffix reversal over the whole permutation, which means that the cost to partition the permutation using our algorithm is $O(n \lg n)$.

Therefore, the cost of WPSR to sort a permutation is $O(n \lg^2 n)$. Along with Lemma 44, this algorithm is a $O(\lg^2 n)$-approximation. Also, note that the time complexity of WPSR is $O(n \lg^2 n)$ (an analysis similar to the one presented on Theorem 7 can be done). □

Algorithm 21 Partition algorithm for WPSR.

PARTITIONWPSR(π, n)

Input: permutation π and its size n
Output: cost used to partition π

Let $x \geq 0$ be the highest integer such that, for all $i \in [1..x]$, $\pi_i \leq \lceil \frac{n}{2} \rceil$ and let $y \leq n'$ be the smallest integer such that, for all $i \in [y..n']$, $\pi_i > \lceil \frac{n}{2} \rceil$
if $x = y - 1$ **then** // Base case: permutation is already partitioned
    return 0
$c \leftarrow 0$
if there is an integer z such that $\pi_i > \lceil \frac{n}{2} \rceil$ and $\pi_j \leq \lceil \frac{n}{2} \rceil$ for all $i \in [x+1..z]$ and $j \in [z+1..y-1]$ **then** // Base case: permutation is partitioned in at most four parts
    if $x > 0$ **then**
        $\pi \leftarrow \pi \cdot \rho_p(z)$
        $c \leftarrow c + f(z)$
    $\pi \leftarrow \pi \cdot \rho_p(y-1)$
    $c \leftarrow c + f(y-1)$
    return c
$c \leftarrow$ PARTITIONWPR($\pi, \lceil \frac{n}{2} \rceil$, INC, $\lceil \frac{n}{2} \rceil$)
        // Partition from 1 to $\lceil \frac{n}{2} \rceil$ with INC type considering pivot $\lceil \frac{n}{2} \rceil$
$c \leftarrow c +$ PARTITIONWSR($\pi, \lceil \frac{n}{2} \rceil + 1$, DEC, $\lceil \frac{n}{2} \rceil$)
        // Partition from $\lceil \frac{n}{2} \rceil + 1$ to n with DEC type considering pivot $\lceil \frac{n}{2} \rceil$
Let j and k be the first and the last position, respectively, of the interval that contains only elements greater than $\lceil \frac{n}{2} \rceil$
if $k < n$ **then**
    $\pi \leftarrow \pi \cdot \rho_s(j)$
    $c \leftarrow c + f(n - j + 1)$
return c

Similarly to what was done regarding SBBWPR, we can use partitionWPSR as an algorithm for SBBWPSR and it can guarantee an approximation factor of $O(\lg n)$ for this problem, as stated in Theorem 10.

Theorem 10. *For $\alpha = 1$,* SBBWPSR *is $O(\lg n)$-approximable.*

Proof. This proof is similar to the one of Theorem 8. □

6.1.3 Sorting by Length-Weighted Prefix Transpositions and Sorting by Length-Weighted Prefix and Suffix Transpositions

We can also adapt `WPR` to use only prefix transpositions and make an algorithm for SbWPT. When prefix reversals are allowed, a decreasing interval can be turned into an increasing interval with one rearrangement, which cannot be done with prefix transpositions only. Therefore, the main difference is that the sorting algorithm does not need a "type" anymore, because it only tries to sort the intervals in increasing order. This means that the partition algorithm will always leave the elements that are greater than the median at the beginning and the elements less than or equal to the median at the end. We then must sort the first half in increasing order, exchange it with the second half, and also sort it in increasing order. Algorithms 22 and 23 show the sorting (`WPT`) and the partition (`partitionWPT`) algorithms for SbWPT, respectively.

Algorithm 22 An $O(\lg^2 n)$-approximation algorithm for SbWPT.

WPT(π, n')
Input: permutation π and integer n'
Output: cost to sort π from position 1 to n'
if $\pi_1, \pi_2, \ldots, \pi_{n'}$ is sorted in increasing form **then**
  return 0
if $n' \leq 1$ **then**
  return 0
if $n' = 2$ **then**
  if $\pi_1 > \pi_2$ **then**
    $\pi \leftarrow \pi \cdot \tau_p(2, 3)$
    return $f(2)$
  return 0
$m \leftarrow \min\{\pi_1, \pi_2, \ldots, \pi_{n'}\} - 1 + \lceil n'/2 \rceil$
$c \leftarrow$ PARTITIONWPT(π, n', m) // Partition π regarding the median m
$c \leftarrow c +$ WPT$(\pi, n' - \lceil n'/2 \rceil)$ // Sort from 1 to $n' - \lceil n'/2 \rceil$ (elements greater than m)
$\pi \leftarrow \pi \cdot \tau_p(n' - \lceil n'/2 \rceil + 1, n' + 1)$
$c \leftarrow c + f(n')$
$c \leftarrow c +$ WPT$(\pi, \lceil n'/2 \rceil)$ // Sort from 1 to $\lceil n'/2 \rceil$
return c

Again, we can adapt `WPT` to use only suffix transpositions and use both to make an algorithm for SbWPST. With the same idea we used for `WPSR`, we can create `WPST` and `partitionWPST`.

Theorems 11 and 12 show that the approximation factor of algorithms `WPT` and `WPST` is $O(\lg^2 n)$.

Theorem 11. *For $\alpha = 1$,* SbWPT *is $O(\lg^2 n)$-approximable.*

Proof. According to Algorithm 22, it is easy to see that $\texttt{WPT}(\pi, n) \leq 2 \times \texttt{WPT}(\pi, n/2) + \texttt{partitionWPT}(\pi, n) + O(n)$. Also, $\texttt{partitionWPT}(\pi, n) \leq 2 \times \texttt{partitionWPT}(\pi, n/2) + O(n)$, which leads us to $\texttt{partitionWPT}(\pi, n)$ being $O(n \lg n)$ and $\texttt{WPT}(\pi, n)$ being $O(n \lg^2 n)$. Using Lemma 43, the approximation factor of `WPT` is $O(n \lg^2 n)/n$, which is $O(\lg^2 n)$. □

Theorem 12. *For $\alpha = 1$,* SBWPST *is $O(\lg^2 n)$-approximable.*

Proof. This proof can be done by using similar arguments as in proof of Theorem 9. □

Algorithm 23 Partition algorithm for `WPT`.

PARTITIONWPT(π, n', m)

Input: permutation π, integer n', and integer m
Output: cost to partition interval from 1 to n' of π according to pivot m

if $n' \leq 1$ **then**
  return 0
Let x be the highest integer such that, for all $i \in [1..x]$, $\pi_i > m$
Let y be the smallest integer such that, for all $i \in [y..n']$, $\pi_i \leq m$
if $x = y - 1$ **then**
  return 0
if there is an integer z such that $\pi_i \leq m$ for all $i \in [x+1..z]$ and $\pi_j > m$ for all $j \in [z+1..y-1]$ **then**
  $\pi \leftarrow \pi \cdot \tau_p(z+1, y)$
  return $f(y-1)$
$n' \leftarrow y - 1$
$c \leftarrow$ PARTITIONWPT$(\pi, \left\lceil \frac{n'}{2} \right\rceil, m)$
$\pi \leftarrow \pi \cdot \tau_p(\left\lceil \frac{n'}{2} \right\rceil + 1, n' + 1)$
$c \leftarrow c + f(n')$
$c \leftarrow c +$ PARTITIONWPT$(\pi, n' - \left\lceil \frac{n'}{2} \right\rceil, m)$
if there are two intervals that contain only elements greater than m, with j (resp. k) being the first (resp. last) position of the second interval **then**
  $\pi \leftarrow \pi \cdot \tau_p(j, k+1)$
  $c \leftarrow c + f(k)$
return c

Similarly to what was done regarding SBBWPR, we can use both `partitionWPT` and `partitionWPST` as algorithms for SBBWPT and SBBWPST, respectively, and they can guarantee an approximation factor of $O(\lg n)$ for these problems, as stated in Theorem 13.

Theorem 13. *For $\alpha = 1$,* SBBWPT *and* SBBWPST *are $O(\lg n)$-approximable.*

Proof. This proof is similar to the one of Theorem 8. □

6.1.4 Sorting by Length-Weighted Prefix Reversals and Transpositions and Sorting by Length-Weighted Prefix and Suffix Reversals and Transpositions

For SBWPRT, the two rearrangements available act only over the prefix of the permutation. Therefore, we designed a slightly different approach, although the idea is still very similar to the previous one. First note that `partitionWPR` has two important parts when acting on an interval from 1 to n' of a permutation π: (i) a call to `partitionWPR`$(\pi, \left\lceil \frac{n'}{2} \right\rceil, 1 - type, m)$ followed by $\rho_p(n')$; and (ii) a call to `partitionWPR`$(\pi, n' - \left\lceil \frac{n'}{2} \right\rceil, 1 - type, m)$ followed by a prefix reversal that finishes the partition. In each of these parts, we have

a moment when we perform one rearrangement, which in the case of SbWPRT could be adapted so that either a prefix reversal or a prefix transposition could be used. Because of this, there are four possibilities for `partitionWPRT`:

1. a call to `partitionWPRT`$(\pi, \lceil \frac{n'}{2} \rceil, type, m)$ followed by $\tau_p(\lceil \frac{n'}{2} \rceil + 1, n' + 1)$; and (ii) a call to `partitionWPRT`$(\pi, n' - \lceil \frac{n'}{2} \rceil, type, m)$ followed by a prefix transposition to finish the partition;

2. a call to `partitionWPRT`$(\pi, \lceil \frac{n'}{2} \rceil, type, m)$ followed by $\tau_p(\lceil \frac{n'}{2} \rceil + 1, n' + 1)$; and (ii) a call to `partitionWPRT`$(\pi, n' - \lceil \frac{n'}{2} \rceil, 1 - type, m)$ followed by a prefix reversal to finish the partition;

3. a call to `partitionWPRT`$(\pi, \lceil \frac{n'}{2} \rceil, 1 - type, m)$ followed by $\rho_p(n')$; and (ii) a call to `partitionWPRT`$(\pi, n' - \lceil \frac{n'}{2} \rceil, type, m)$ followed by a prefix transposition to finish the partition;

4. a call to `partitionWPRT`$(\pi, \lceil \frac{n'}{2} \rceil, 1 - type, m)$ followed by $\rho_p(n')$; and (ii) a call to `partitionWPRT`$(\pi, n' - \lceil \frac{n'}{2} \rceil, 1 - type, m)$ followed by a prefix reversal to finish the partition.

Note that depending on which rearrangement is performed, the recursive calls vary using *type* or $1 - type$, where *type* can be INC$= 1$ or DEC$= 0$. Also, note that any of these possibilities can be used, but the experimental results (Section 6.5) indicates that the second one performs better than the others.

For the main algorithm, `WPRT`, although we could use a prefix transposition between its recursive calls, we decided to keep it exactly as `WPR`, only adapting it to use `partitionWPRT`.

In a similar manner, we have four options for the partition when only suffix reversals and suffix transpositions are allowed. With this partition algorithm, the sorting variation that uses only suffix reversals and transpositions, and the algorithms for SbWPRT, we can make an algorithm for SbWPSRT (called `WPSRT`) in the same way we did for SbWPSR and SbWPST: explicitly partition the permutation, if it is not already separated as stated in Definition 22, and then use the prefix algorithm to sort the beginning and the suffix algorithm to sort the end of the permutation.

Regardless of which of the four possibilities is chosen for `partitionWPRT` or its suffix version, the algorithms for SbWPRT and for SbWPSRT are also $O(\lg^2 n)$-approximations, as shown in Theorems 14 and 15.

Theorem 14. *For $\alpha = 1$,* SbWPRT *is $O(\lg^2 n)$-approximable.*

Proof. Note that all the rearrangements performed by `WPRT` or `partitionWPRT`, regardless of which of the four possibilities of the partition is used, have the same cost of $O(n)$, where n is the amount of valid elements. Therefore, the analysis for this problem is similar to the analysis for `WPR` or `WPT`, which are shown by Theorems 7 and 11. □

Theorem 15. *For $\alpha = 1$,* SbWPSRT *is $O(\lg^2 n)$-approximable.*

Proof. This proof is similar to the ones of Theorems 9, 12, and 14. □

Again, we can use `partitionWPRT` and `partitionWPSRT` to sort binary strings and guarantee an approximation factor of $O(\lg n)$ for these problems, as stated in Theorem 16.

Theorem 16. *For $\alpha = 1$,* SBBWPRT *and* SBBWPSRT *are $O(\lg n)$-approximable.*

Proof. This proof is similar to the one for Theorem 8. □

6.2 Bounds on the Diameters

We divided this section in two main parts, in order to show how to obtain lower bounds for the diameters and then upper bounds for them. In the second part, we use the algorithms defined in Section 6.1.

6.2.1 Lower Bounds on the Diameters

In this section we show bounds to transform the binary string $T' = 0101...01$ into $T'_s = 0^{n/2}1^{n/2}$ (that is, to sort T'), in order to give lower bounds on $C^\alpha_\beta(n)$ and, consequently, on $D^\alpha_\beta(n)$.

6.2.1.1 Considering $0 < \alpha < 1$

Bender *et al.* [5] showed that $C^\alpha_r(n)$ and $D^\alpha_r(n)$ are in $\Omega(n)$. The fact that $C^\alpha_r(n) \leq C^\alpha_{pr}(n)$ and $D^\alpha_r(n) \leq D^\alpha_{pr}(n)$ leads directly to Lemma 48.

Lemma 48. *For $0 < \alpha < 1$, $C^\alpha_{pr}(n)$ and $D^\alpha_{pr}(n)$ are $\Omega(n)$.*

For any binary string $T = t_1 t_2 \dots t_n$, let $P_{0<\alpha<1}(T) = \sum_{i=1}^{n-1} |t_i - t_{i+1}|$. Lemma 49 gives a lower bound on $C^\alpha_{pt}(n)$ and $D^\alpha_{pt}(n)$.

Lemma 49. *For $0 < \alpha < 1$, $C^\alpha_{pt}(n)$ and $D^\alpha_{pt}(n)$ are $\Omega(n)$.*

Proof. It is easy to see that $P_{0<\alpha<1}(T') = n - 1$ and $P_{0<\alpha<1}(T'_s) = 1$. Since a prefix transposition changes $P_{0<\alpha<1}(T)$ by at most 2, at least $((n-1)-1)/2 = n/2 - 1$ of them are needed to sort T'. Since any transposition has length greater than or equal to 2, the cost to sort T' is at least $\Omega(2^\alpha n) = \Omega(n)$. □

The arguments given in Lemma 49 can be directly used for the case when prefix reversals are allowed along with prefix transpositions (because a prefix reversal can change $P_{0<\alpha<1}(T)$ by at most 1). This leads to Lemma 50.

Lemma 50. *For $0 < \alpha < 1$, $C^\alpha_{prt}(n)$ and $D^\alpha_{prt}(n)$ are $\Omega(n)$.*

6.2.1.2 Considering $\alpha = 1$

Bender *et al.* [5] showed that $C^\alpha_r(n)$ and $D^\alpha_r(n)$ are in $\Omega(n \lg n)$. The fact that $C^\alpha_r(n) \leq C^\alpha_{pr}(n)$ and $D^\alpha_r(n) \leq D^\alpha_{pr}(n)$ leads directly to Lemma 51.

Lemma 51. *For $\alpha = 1$, $C^\alpha_{pr}(n)$ and $D^\alpha_{pr}(n)$ are $\Omega(n \lg n)$.*

Given any binary string T that has the same amount of 0's and 1's, match the kth 0 with the kth 1, for $1 \leq k \leq n/2$, and keep this matching during the sorting process of such sequence. Let T^* be a binary string that can be generated from T by applying on it some allowed prefix rearrangements. We define the *separation measure* $d(T, T^*, k)$ between the kth 0 and the kth 1 of T as the number of bits between them plus one unit considering their positions in T^*.

Example 23. *Let* $T = 0001011110$ *and* $T^* = T \cdot \tau_p(3, 7) = 0101001110$. *Due to the matching we make in* T, *we can rewrite them as* $T = 0_1 0_2 0_3 1_1 0_4 1_2 1_3 1_4 1_5 0_5$ *and* $T^* = 0_3 1_1 0_4 1_2 0_1 0_2 1_3 1_4 1_5 0_5$ *in order to facilitate the computation of the separation measures. Thus, we have* $d(T, T^*, 1) = 3$, $d(T, T^*, 2) = 2$, $d(T, T^*, 3) = 6$, $d(T, T^*, 4) = 5$, *and* $d(T, T^*, 5) = 1$.

Now let $P_{\alpha=1}(T, T^*) = \sum_{k=1}^{n/2} \lg d(T, T^*, k)$. Lemma 54 gives a lower bound on $C^{\alpha}_{pt}(n)$ and $C^{\alpha}_{prt}(n)$, and, consequently, on $D^{\alpha}_{pt}(n)$ and $D^{\alpha}_{prt}(n)$. It uses Lemmas 52 and 53 presented next.

Lemma 52. $P_{\alpha=1}(T', T'_s) = \Omega(n \lg n)$.

Proof. In T'_s, the distance from each of the $n/4$ first 0's to its matched 1 is at least $n/4$. Therefore,

$$P_{\alpha=1}(T', T'_s) \geq \sum_{k=1}^{n/4} \lg \frac{n}{4} \geq \frac{n}{4} \lg \frac{n}{4},$$

which is $\Omega(n \lg n)$. □

Lemma 53. *Let* λ *be a prefix reversal or a prefix transposition of length* ℓ. *For any binary string* T *with the same amount of* 0*'s and* 1*'s and* T^* *that can be generated from* T *by applying some allowed prefix rearrangements on it, we have that* $P_{\alpha=1}(T, T^* \cdot \lambda) - P_{\alpha=1}(T, T^*) \leq q\ell$, *where* q *is a constant.*

Proof. Let $T^* = t_1\ t_2\ \ldots\ t_n$. Note that if λ is a prefix reversal $\rho_p(i)$, then $d(T, T^* \cdot \rho_p(i), k)$ only changes if the position of one element of the kth 0/1 pair is less than or equal to i and the position of the other element of the pair is greater than i. A prefix transposition $\tau_p(i, j)$, however, exchanges two segments of the sequence, which means that $d(T, T^* \cdot \tau_p(i, j), k)$ can change in two manners: (1) one element of the kth 0/1 pair is in a position less than j and the other is in a position greater than or equal to j, or (2) one element of the kth 0/1 pair is in a position less than i and the other is between i and $j - 1$.

Assume $\lambda = \tau_p(i, j)$ first and consider that case (1) above happens. Without loss of generality, choose some k where the 0 appears inside the prefix transposition (before position j) and the 1 appears outside it (after or at position j) in T^*, and denote as x the distance between the 1 and position $j - 1$:

$$T^* = \underbrace{t_1\ t_2\ \ldots\ 0\ \ldots\ t_{i-1} t_i\ t_{i+1}\ \ldots\ t_{j-1}\ \overbrace{t_j\ \ldots\ 1}^{x}}_{d(T,T^*,k)}\ \ldots\ t_n$$

Note that $x \leq d(T, T^*, k)$. Also, note that $d(T, T^* \cdot \lambda, k) \leq d(T, T^*, k) + \ell$, because then 0 can be placed at most ℓ bits away from where it is in T^*. Therefore, the contribution of the distance of this pair to the function $P_{\alpha=1}$ is less than $\lg(\ell + d(T, T^*, k)) - \lg(d(T, T^*, k)) = \lg(1 + \ell/d(T, T^*, k)) \leq \lg(1 + \ell/x)$.

Now consider that case (2) happens. Again, without loss of generality, assume that the 0 appears inside the first segment of the prefix transposition (before position i) and the 1 appears in the second segment (between positions i and $j-1$) in T^*, and assume that x is the distance between the 1 and position $i-1$:

$$T^* = \underbrace{t_1\ t_2\ \ldots\ 0\ \ldots\ t_{i-1}\ \overbrace{t_i\ t_{i+1}\ \ldots\ 1}^{x}\ \ldots\ t_{j-1}}_{d(T,T^*,k)}\ t_j\ \ldots\ 1\ \ldots\ t_n$$

Note that $x \leq d(T, T^*, k)$ and that $d(t, T^* \cdot \lambda, k) \leq d(T, T^*, k) + \ell$, which can be tight if we consider both elements of the pair as close to position i as possible. Therefore, the contribution of the distance of this pair to the function $P_{\alpha=1}$ is less than $\lg(\ell + d(T, T^*, k)) - \lg(d(T, T^*, k)) = \lg(1 + \ell/d(T, T^*, k)) \leq \lg(1 + \ell/x)$.

Now we have two important facts. The first one is that the maximum change that can occur in the function $P_{\alpha=1}$ will happen if all ℓ bits of the segment affected by the rearrangement contribute to the change. Second, note that the more x decreases, the higher $\lg(1 + \ell/x)$ gets: for case (1) this means that the bit that is outside the prefix transposition has to be as close as possible to position j while for case (2) this means that the bit in the second segment has to be as close as possible to position i.

Assume then that t of the ℓ bits are of case (1) and that the other $\ell - t$ are of case (2). This means that the highest change in $P_{\alpha=1}$ will have $(\ell - t)/2$ bits of case (2) in each segment of the prefix transposition. According to all these facts, we have that

$$P_{\alpha=1}(T, T^* \cdot \tau_p(i,j)) - P_{\alpha=1}(T, T^*) \leq \sum_{j=1}^{t} \lg\left(1 + \frac{\ell}{j}\right) + \sum_{j=1}^{(\ell-t)/2} \lg\left(1 + \frac{\ell}{j}\right).$$

It is easy to see that the right side of the expression above is maximum when $t = \ell$, which leads us to

$$\begin{aligned} P_{\alpha=1}(T, T^* \cdot \tau_p(i,j)) - P_{\alpha=1}(T, T^*) &\leq \textstyle\sum_{j=1}^{\ell} \lg\left(1 + \frac{\ell}{j}\right) \\ &\leq \textstyle\sum_{j=1}^{\ell} \left(1 + \lg\left(\frac{\ell}{j}\right)\right) \\ &= \ell + \textstyle\sum_{j=1}^{\ell} \lg\left(\frac{\ell}{j}\right) \\ &= \ell + \lg\left(\frac{\ell^\ell}{\ell!}\right) \leq \ell + \lg \mathrm{e}^\ell = (1 + \lg \mathrm{e})\ell. \end{aligned}$$

When $\lambda = \rho_p$, the proof is very similar, being sufficient to use the same considerations made for case (1). □

Lemma 54. *For $\alpha = 1$, $C^\alpha_{pt}(n)$, $D^\alpha_{pt}(n)$, $C^\alpha_{prt}(n)$, and $D^\alpha_{prt}(n)$ are $\Omega(n \lg n)$.*

Proof. Note that $P_{\alpha=1}(T', T') = 0$ and $P_{\alpha=1}(T', T'_s) = \Omega(n \lg n)$ (see Lemma 52). Also, a prefix transposition or a prefix reversal of length ℓ increases $P_{\alpha=1}$ by at most $q\ell$ where

$q = O(1)$ (see Lemma 53).

Consider an optimal sequence that sorts T' and contains d rearrangements (prefix transpositions or prefix reversals) $\lambda_1, \lambda_2, \ldots, \lambda_d$ of lengths $\ell_1, \ell_2, \ldots, \ell_d$. We know that $P_{\alpha=1}(T', T' \cdot \lambda_1) - P_{\alpha=1}(T', T') \leq q\ell_1$, $P_{\alpha=1}(T', T' \cdot \lambda_1 \cdot \lambda_2) - P_{\alpha=1}(T', T' \cdot \lambda_1) \leq q\ell_2$, and so on, until $P_{\alpha=1}(T', T'_s) - P_{\alpha=1}(T', T' \cdot \lambda_1 \cdots \lambda_{d-1}) \leq q\ell_d$. All these together show that

$$P_{\alpha=1}(T', T'_s) - P(T', T') \leq \sum_{i=1}^{d} q\ell_i = q \sum_{i=1}^{d} \ell_i = q \sum_{i=1}^{d} f(\ell_i) = q b^{\alpha}_{\beta}(T'),$$

for $\beta \in \{\tau_p, \rho_p\tau_p\}$.

Therefore, $c^{\alpha}_{\beta}(T') \geq (P_{\alpha=1}(T', T'_s) - P_{\alpha=1}(T', T'))/q = (\Omega(n \lg n) - 0)/q$, which means that $c^{\alpha}_{\beta}(T')$ is $\Omega(n \lg n)$ and, thus, $C^{\alpha}_{\beta}(n) = \Omega(n \lg n)$ and $D^{\alpha}_{\beta}(n) = \Omega(n \lg n)$. □

6.2.1.3 Considering $\alpha > 1$

From Lemma 43 we know that $d^{\alpha}_{\beta}(\pi) \geq n^{\alpha}$ and $c^{\alpha}_{\beta}(T) \geq n^{\alpha}$ for $\beta \in \{pr, pt, prt\}$. This leads directly to Lemma 55.

Lemma 55. *For $\alpha = 1$, $C^{\alpha}_{pr}(n)$, $D^{\alpha}_{pr}(n)$, $C^{\alpha}_{pt}(n)$, $D^{\alpha}_{pt}(n)$, $C^{\alpha}_{prt}(n)$, and $D^{\alpha}_{prt}(n)$ are $\Omega(n^{\alpha})$.*

6.2.2 Upper Bounds on the Diameters

The algorithms presented in Section 6.1 actually can sort permutations and binary strings for any $\alpha \geq 0$. Therefore, they can give upper bounds on the diameters, as we show next.

Let $S_{\beta}(n)$ (resp. $P_{\beta}(n)$) be the cost used by the sorting (resp. partition) algorithm to sort (resp. partition) any permutation with n valid elements. It is easy to see that $D^{\alpha}_{\beta}(n) \leq S_{\beta}(n)$. We also have that $C^{\alpha}_{\beta}(n) \leq P_{\beta}(n)$, because the partition algorithm for a permutation π can be used as a sorting algorithm for any binary string $T = M(n, p, \pi)$ for any pivot $0 \leq p \leq n$, even though the sorting algorithms use the partition algorithms only to sort $T = M(n, \lceil n/2 \rceil, \pi)$. Lemma 56 shows upper bounds on the diameter for the problems of sorting permutations or binary strings by length-weighted prefix rearrangements.

Lemma 56. *For $\beta \in \{pr, pt, prt\}$,*

$$P_{\beta}(n) \text{ and } C^{\alpha}_{\beta}(n) \text{ are } \begin{cases} O(n) & \text{if } 0 < \alpha < 1 \\ O(n \lg n) & \text{if } \alpha = 1 \\ O(n^{\alpha}) & \text{if } \alpha > 1. \end{cases} \tag{6.2}$$

and

$$S_{\beta}(n) \text{ and } D^{\alpha}_{\beta}(n) \text{ are } \begin{cases} O(n \lg n) & \text{if } 0 < \alpha < 1 \\ O(n \lg^2 n) & \text{if } \alpha = 1 \\ O(n^{\alpha}) & \text{if } \alpha > 1. \end{cases} \tag{6.3}$$

Proof. From the algorithms given in Section 6.1, we can see that

$$P_{\beta}(n) \leq 2P_{\beta}\left(\frac{n}{2}\right) + 2n^{\alpha} \tag{6.4}$$

and

$$S_\beta(n) \leq 2S_\beta\left(\frac{n}{2}\right) + P_\beta(n) + n^\alpha. \tag{6.5}$$

Using the Master Theorem [19, Sec. 4.5] over Equations (6.4) and (6.5), we can easily find the stated upper bounds. □

The lower bounds given in Section 6.2.1 along with the upper bounds given in Lemma 56 directly give us the following theorems.

Theorem 17. *For* $\beta \in \{pr, pt, prt\}$,

$$C_\beta^\alpha(n) \text{ is } \begin{cases} \Theta(n) & \text{if } 0 < \alpha < 1 \\ \Theta(n \lg n) & \text{if } \alpha = 1 \\ \Theta(n^\alpha) & \text{if } \alpha > 1. \end{cases} \tag{6.6}$$

Theorem 18. *For* $\beta \in \{pr, pt, prt\}$ *and* $\alpha > 1$, $D_\beta^\alpha(n)$ *is* $\Theta(n^\alpha)$.

6.3 Sorting Algorithms Considering $0 < \alpha < 1$

As we already mentioned, the partition algorithms for prefix rearrangements described in Section 6.1 for $\alpha = 1$ can be used to sort binary strings and guarantee approximation factors of $O(\lg n)$ in this case. Now we show, in Theorem 19, that they can also guarantee an approximation factor of $O(\lg n)$ when $0 < \alpha < 1$.

Theorem 19. *For* $0 < \alpha < 1$, *problems* SBbWPR, SBbWPT, SBbWPRT *are* $O(\lg n)$-*approximable.*

Proof. Let $T = 0^{w_0}1^{w_1}\ldots 0^{w_{2g}}1^{w_{2g+1}}$ be a binary string. Create string T' by mapping each block of T (of 0's or 1's) into one element in T' (a 0 or a 1, accordingly). Now, we use `partitionWPR`, `partitionWPT`, or `partitionWPRT` (depending on the rearrangement model) to sort T' and then we map back each rearrangement, according to the initial mapping, in order to sort T. Note that none of these algorithms will consider the last block of 1's (because the goal is to obtain 1's at the end of the string).

The first important observation is that each element of T' takes part in at most $2\lg n$ rearrangements in any of the algorithms. This happens because each recursive step contains two rearrangements and there are at most $\lg n$ recursive steps.

Now consider a prefix reversal or a prefix transposition of length ℓ that contains the q first blocks of T. So, $\ell = w_0 + w_1 + \ldots + w_q$. Since $0 < \alpha < 1$, we have that $\ell^\alpha \leq w_0^\alpha + w_1^\alpha + \cdots + w_q^\alpha = \sum_{i=0}^{q} w_i^\alpha$. Since $q \leq 2g$, the total cost used by any of these algorithms is at most

$$\left(\sum_{i=0}^{2g} w_i^\alpha\right) \times (2\lg n).$$

We defined, for Lemma 45, that $P(T) = (\sum_{i=0}^{2g} w_i^\alpha)/4$, and we showed that $c_\beta^\alpha(T) \geq P(T)$ for $\beta \in \{\rho_p, \tau_p, \rho_p\tau_p\}$. Therefore, the total cost of any of the algorithms is at most

$$(4P(T)) \times (2\lg n) = 8\lg n P(T) \leq 8\lg n c_\beta^\alpha(T),$$

which indicates that they are $O(\lg n)$-approximations. □

Consider again algorithm `3-PR` for SbPR described in the beginning of Section 6.1 (Algorithm 13). Such algorithm is a $O(n^\alpha)$-approximation for SbWPR when $0 < \alpha < 1$, as Theorem 20 shows.

Theorem 20. *For $0 < \alpha < 1$,* SbWPR *is $O(n^\alpha)$-approximable.*

Proof. Note that algorithm `3-PR` performs at most $b_{upr}(\pi)$ steps and each step contains at most three prefix reversals. As a high estimative, we could say that each prefix reversal has length n, thus costing n^α. Therefore, such algorithm uses a cost of at most $3n^\alpha b_{upr}(\pi)$ to sort a permutation using only prefix reversals. The lower bound given by Lemma 46 shows that

$$3n^\alpha b_{upr}(\pi) \leq 3n^\alpha \left(\frac{d_{pr}^\alpha(\pi)}{2^\alpha} \right) \leq \frac{3n^\alpha}{2^\alpha} d_{pr}^\alpha(\pi),$$

and we can see that it is a $(3/2^\alpha)n^\alpha$-approximation. □

We can adapt algorithm `3-PR` for SbWPT and SbWPRT (see Algorithms 24 and 25) in order to also have $O(n^\alpha)$-approximations for both problems, as Theorem 21 shows.

Algorithm 24 Algorithm `3-PR` adapted for SbWPT.

3-PR-FOR-SBWPT(π, n)

Input: permutation π and its size n
Output: cost used to sort π

```
1  c ← 0
2  while π ≠ ι_n do
3      Let i be the highest element such that π_i ≠ i
4      Let ℓ be the position of the last element of the strip that contains i
5      π ← π · τ_p(ℓ + 1, i + 1)
6      c ← c + f(i)
7  return c
```

Algorithm 25 Algorithm `3-PR` adapted for SbWPRT.

3-PR-FOR-SBWPRT(π, n)

Input: permutation π and its size n
Output: cost used to sort π

```
1  c ← 0
2  while π ≠ ι_n do
3      Let i be the highest element such that π_i ≠ i
4      Let b be the position of the first element of the strip that contains i
5      Let e be the position of the last element of the strip that contains i
6      if π_b = i then      // If the strip is decreasing, transform it into an increasing one
7          π ← π · ρ_p(e)
8          c ← c + f(e); e ← b − e + 1; b ← 1
9      π ← π · τ_p(b − e + 2, i + 1)
10     c ← c + f(i)
11 return c
```

Theorem 21. *For* $0 < \alpha < 1$, SBWPT *and* SBWPRT *are* $O(n^\alpha)$*-approximable.*

Proof. In the case of SBWPT, since the concept of decreasing strip does not exist, it is possible to put the strip that contains the highest element out of order in its correct position with only one prefix transposition, as Algorithm 24 shows. Therefore, such algorithm would use a cost of at most $n^\alpha b_p(\pi)$ to sort a permutation. Due to the lower bound given by Lemma 46, we can see that this is a $(2/2^\alpha)n^\alpha$-approximation.

When both prefix reversals and prefix transpositions are allowed, decreasing strips can exist. Therefore, if the strip that contains the highest element that is out of order is increasing or it is a singleton, then the algorithm can use one prefix transposition to put it in the correct position. However, if the strip is decreasing, then the algorithm must reverse it first, and then put it into the correct position, which means that at most two rearrangements will be used. Algorithm 25 shows this behavior. Therefore, at most a cost of $2n^\alpha b_{upr}(\pi)$ will be used by this algorithm to sort π, and due to the lower bound given by Lemma 46, we can see that it is a $(4/2^\alpha)n^\alpha$-approximation. □

6.4 Sorting Algorithms Considering $\alpha > 1$

In this section we show, in Theorem 22, that the algorithms given in Section 6.1 are $O(1)$-approximations for sorting permutations or binary strings by length-weighted prefix rearrangements for $\alpha > 1$.

Theorem 22. *For* $\alpha > 1$, SBWPR, SBWPT, SBWPRT, SBBWPR, SBBWPT, *and* SBBWPRT *are* $O(1)$*-approximable.*

Proof. Directly from the lower bounds given by Lemma 43 and the upper bounds given by Lemma 56. □

We would like now to determine more precisely this constant approximation factor. Consider once again $S_\beta(n)$ (resp. $P_\beta(n)$) as the cost used by the sorting (resp. partition) algorithm to sort any permutation (resp. partition any permutation or sort any binary string) with n valid elements. From Lemma 56, we know that $P_\beta(n)$ and $S_\beta(n)$ are $O(n^\alpha)$. Therefore, we may assume that $P_\beta(n) \leq f_P n^\alpha$ and $S_\beta(n) \leq f_S n^\alpha$ for some constants f_P and f_S. Due to the lower bound of n^α, it is easy to see that f_P is the approximation factor for partitioning permutations/sorting binary strings and f_S is the approximation factor for sorting permutations.

We start by expanding the recurrences given in Equations (6.4) and (6.5). For the

former, we have

$$\begin{aligned}
P_\beta(n) &\le 2\Big[2P_\beta\Big(\frac{n}{4}\Big)+2\Big(\frac{n}{2}\Big)^\alpha\Big]+2n^\alpha = 4P_\beta\Big(\frac{n}{4}\Big)+4\Big(\frac{n}{2}\Big)^\alpha+2n^\alpha \\
&\le 4\Big[2P_\beta\Big(\frac{n}{8}\Big)+2\Big(\frac{n}{4}\Big)^\alpha\Big]+4\Big(\frac{n}{2}\Big)^\alpha+2n^\alpha \\
&= 8P_\beta\Big(\frac{n}{8}\Big)+8\Big(\frac{n}{4}\Big)^\alpha+4\Big(\frac{n}{2}\Big)^\alpha+2n^\alpha \\
&\ \ \vdots \\
&\le nP_\beta\Big(\frac{n}{n}\Big)+n\Big(\frac{n}{n/2}\Big)^\alpha+\frac{n}{2}\Big(\frac{n}{n/4}\Big)^\alpha+\ldots+4\Big(\frac{n}{4}\Big)^\alpha+2n^\alpha \\
&= 0+\sum_{i=1}^{\lg n}\frac{n}{2^{i-1}}2^{i\alpha} = \frac{2^{\alpha+1}}{2^\alpha-2}(n^\alpha-n) \le \frac{2^{\alpha+1}}{2^\alpha-2}n^\alpha.
\end{aligned} \tag{6.7}$$

Note that $f_P = (2^{\alpha+1})/(2^\alpha-2)$, as defined before, is the approximation factor for sorting binary strings.

For Equation (6.5), we have

$$\begin{aligned}
S_\beta(n) &\le 2\Big[2S_\beta\Big(\frac{n}{4}\Big)+P_\beta\Big(\frac{n}{2}\Big)+\Big(\frac{n}{2}\Big)^\alpha\Big]+P_\beta(n)+n^\alpha \\
&= 4S_\beta\Big(\frac{n}{4}\Big)+\Big[2P_\beta\Big(\frac{n}{2}\Big)+P_\beta(n)\Big]+\Big[2\Big(\frac{n}{2}\Big)^\alpha+n^\alpha\Big] \\
&\ \ \vdots \\
&\le nS_\beta\Big(\frac{n}{n}\Big)+\Big[\frac{n}{2}P_\beta(2)+\frac{n}{4}P_\beta(4)+\ldots+2P_\beta\Big(\frac{n}{2}\Big)+P_\beta(n)\Big] \\
&\quad +\Big[\frac{n}{2}(2)^\alpha+\frac{n}{4}(4)^\alpha+\ldots+2\Big(\frac{n}{2}\Big)^\alpha+n^\alpha\Big] \\
&= 0+\sum_{i=1}^{\lg n}\frac{n}{2^i}P_\beta(2^i)+\sum_{i=1}^{\lg n}\frac{n}{2^i}2^{i\alpha} \\
&= \left(\frac{2^\alpha}{2^\alpha-2}+\frac{2^{2\alpha+1}}{(2^\alpha-2)^2}\right)n^\alpha-\left(\frac{2^\alpha}{2^\alpha-2}+\frac{2^{2\alpha+1}}{(2^\alpha-2)^2}\right)n-\left(\frac{2^{\alpha+1}}{2^\alpha-2}\right)n\lg n \\
&\le \left(\frac{2^\alpha}{2^\alpha-2}+\frac{2^{2\alpha+1}}{(2^\alpha-2)^2}\right)n^\alpha.
\end{aligned} \tag{6.8}$$

As defined before, we take $f_S = 2^\alpha/(2^\alpha-2)+(2^{2\alpha+1})/(2^\alpha-2)^2$, which is the approximation factor for sorting permutations.

From the expressions obtained above, we have the following results, which lead to Corollary 23:

1. If $\alpha \ge 2$, then $f_P \le 4$ and $f_S \le 10$;
2. If $\alpha \ge 3$, then $f_P \le 8/3 < 3$ and $f_S \le 44/9 < 5$;
3. If $\alpha \ge 4$, then $f_P \le 16/7 < 2.5$ and $f_S \le 184/49 < 4$;
4. as α increases, f_P tends to 2 and f_S tends to 3.

Corollary 23. *For $\alpha \ge 2$,* SBWPR*,* SBWPT*, and* SBWPRT *are 10-approximable while* SBBWPR*,* SBBWPT*, and* SBBWPRT *are 4-approximable. For $\alpha \ge 3$,* SBWPR*,* SBWPT*, and* SBWPRT *are 5-approximable while* SBBWPR*,* SBBWPT*, and* SBBWPRT

are 3-approximable. For large values of α, SBWPR, SBWPT, *and* SBWPRT *are* $(3+\epsilon)$-*approximable while* SBBWPR, SBBWPT, *and* SBBWPRT *are* $(2+\epsilon)$-*approximable.*

6.5 Experimental Results

All the algorithms presented in Section 6.1 (for $\alpha = 1$) were implemented in C language and executed over the same sets U1 and U2 introduced in Section 4.7. The experimental results are shown in Figures 6.4 to 6.6. The x-axis represents the values of n while the y-axis represents the average of the approximation factor between the permutations of that size. For Set U1 the approximation factors were calculated using the exact distances of the permutations and for Set U2 they were calculated using the theoretical lower bounds on the distances, which were given in Lemmas 43 and 44.

We point out that we implemented the four possibilities of the partition algorithm for SBWPRT and the results we presented are from the approach that gave better results, which is: a call to `partitionWPRT`$(\pi, \lceil \frac{n'}{2} \rceil, type, m)$ followed by $\tau_p(\lceil \frac{n'}{2} \rceil + 1, n' + 1)$; and (ii) a call to `partitionWPRT`$(\pi, n' - \lceil \frac{n'}{2} \rceil, 1 - type, m)$ followed by a prefix reversal to finish the partition. For the suffix variation the best possibility was the equivalent, that is, we use a suffix transposition after the first recursive call and a suffix reversal after the second call.

Also note that the greedy algorithms mentioned just before the beginning of Section 6.1.1 (those whose names end with "`g`") always performed better than the $O(\lg^2 n)$-approximation algorithms when executed over Set U1. For bigger permutations, however, the contrary clearly happens and our algorithms outperform the others. In particular, note that `WPRm` is the best for permutations of size 2 to 10, but it is by far the worst for long permutations.

Figure 6.7 shows some experimental results for SBWPR when $\alpha \in \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ over Sets U1 and U2 also. Table 6.1 considers some of the values for n that were tested and for each of them we present the worst approximation factors that were calculated for all permutations of that size ($n!$ permutations when $n \leq 10$ and $10,000$ permutations otherwise). The last line of this table shows the theoretical approximation factors, for each value of α, calculated with the formula $2^\alpha/(2^\alpha - 2) + (2^{2\alpha+1})/(2^\alpha - 2)^2$ given in Equation (6.8) of Section 6.4.

The graph that presents the results for Set U1 shows that the average approximation factor tends to be really smaller than the expected for each value of α. More than that, we can see in Table 6.1 that even the worst approximation factors for all $n!$ permutations are still smaller than the expected. Although the graph for Set U2 only presents an average approximation factor for the $10,000$ permutations of each size and this set cannot completely represent the behavior of the algorithms over permutations with big sizes, we can see that the curves tend to stabilize at small values of approximation factors. We can also observe such stabilization in Table 6.1, because the more the permutation size increases, the less the approximation factors increase.

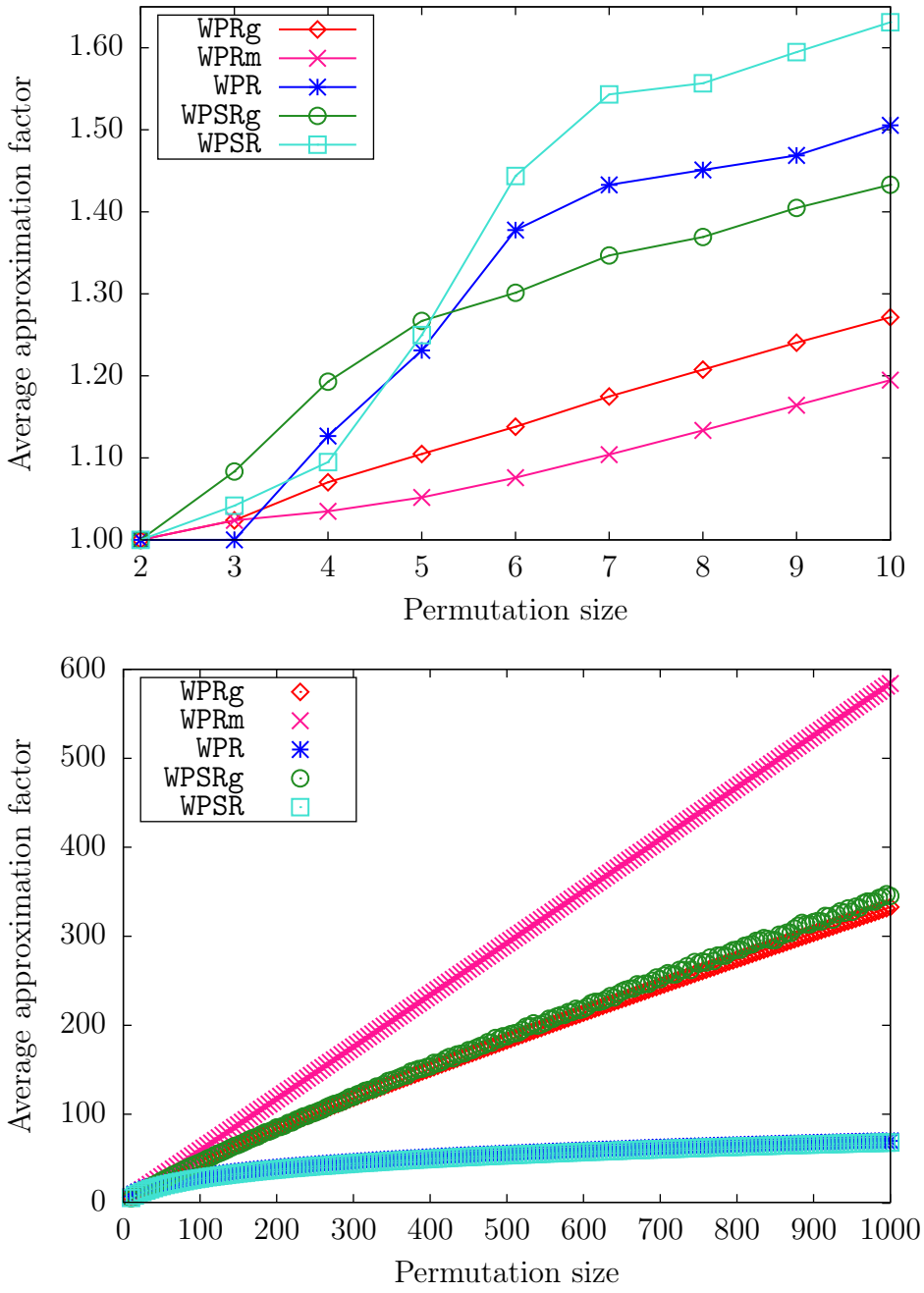


Figure 6.4: Average approximation factors for `WPRg`, `WPRm`, `WPR`, `WPSRg`, and `WPSR` when $\alpha = 1$ and the permutation size grows.

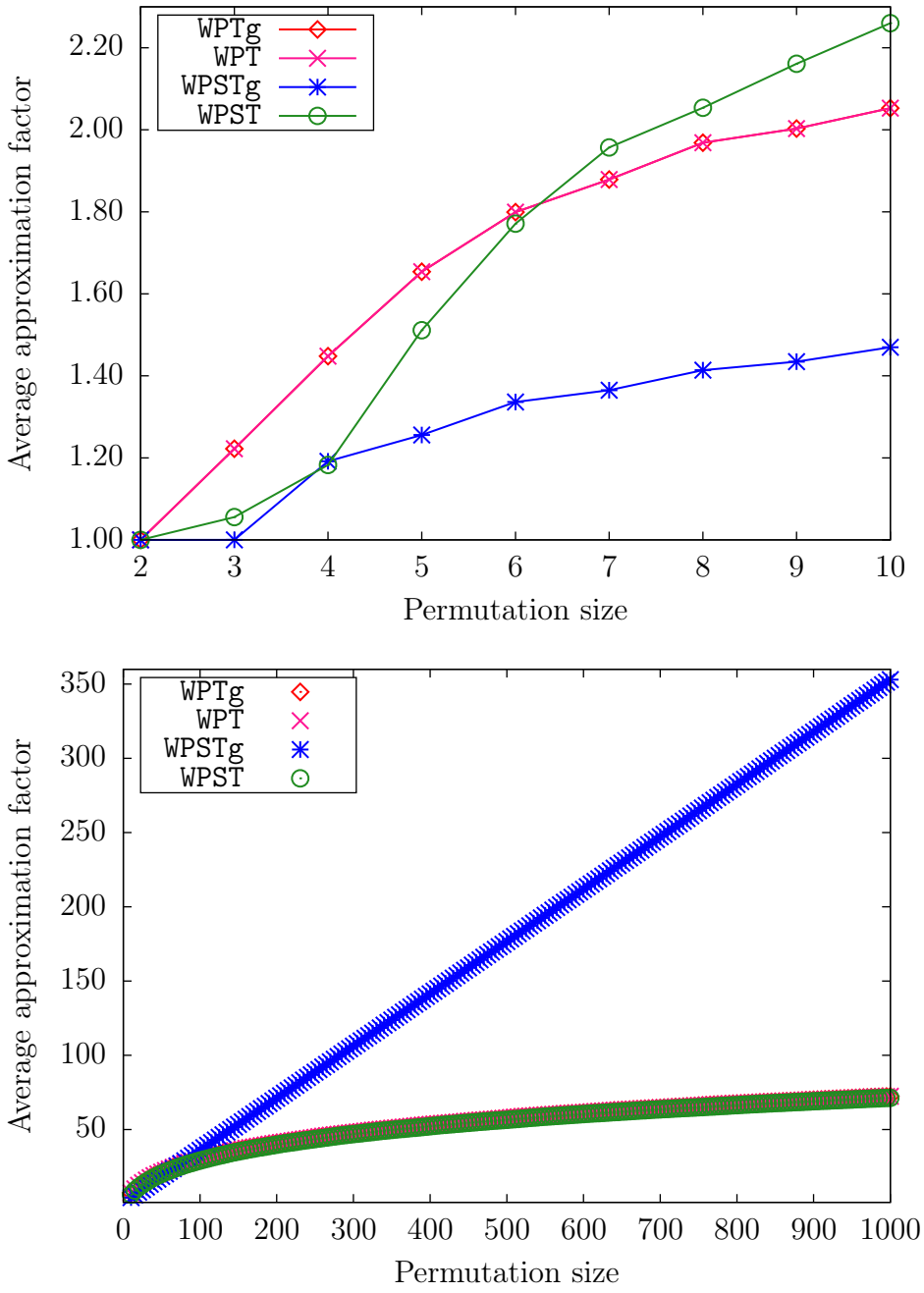


Figure 6.5: Average approximation factors for `WPTg`, `WPT`, `WPSTg`, and `WPST` when $\alpha = 1$ and the permutation size grows.

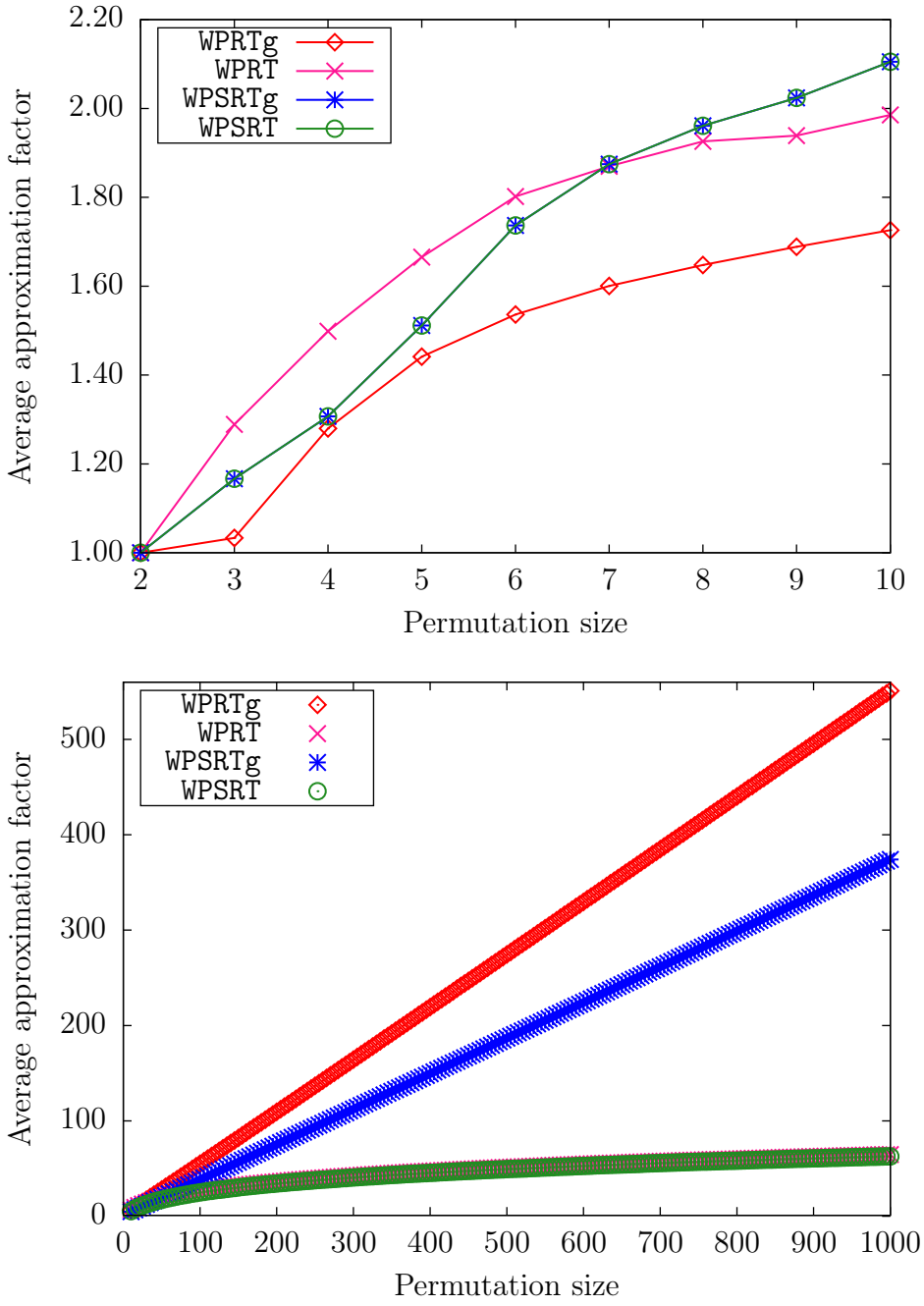


Figure 6.6: Average approximation factors for `WPRTg`, `WPRT`, `WPSRTg`, and `WPSRT` when $\alpha = 1$ and the permutation size grows.

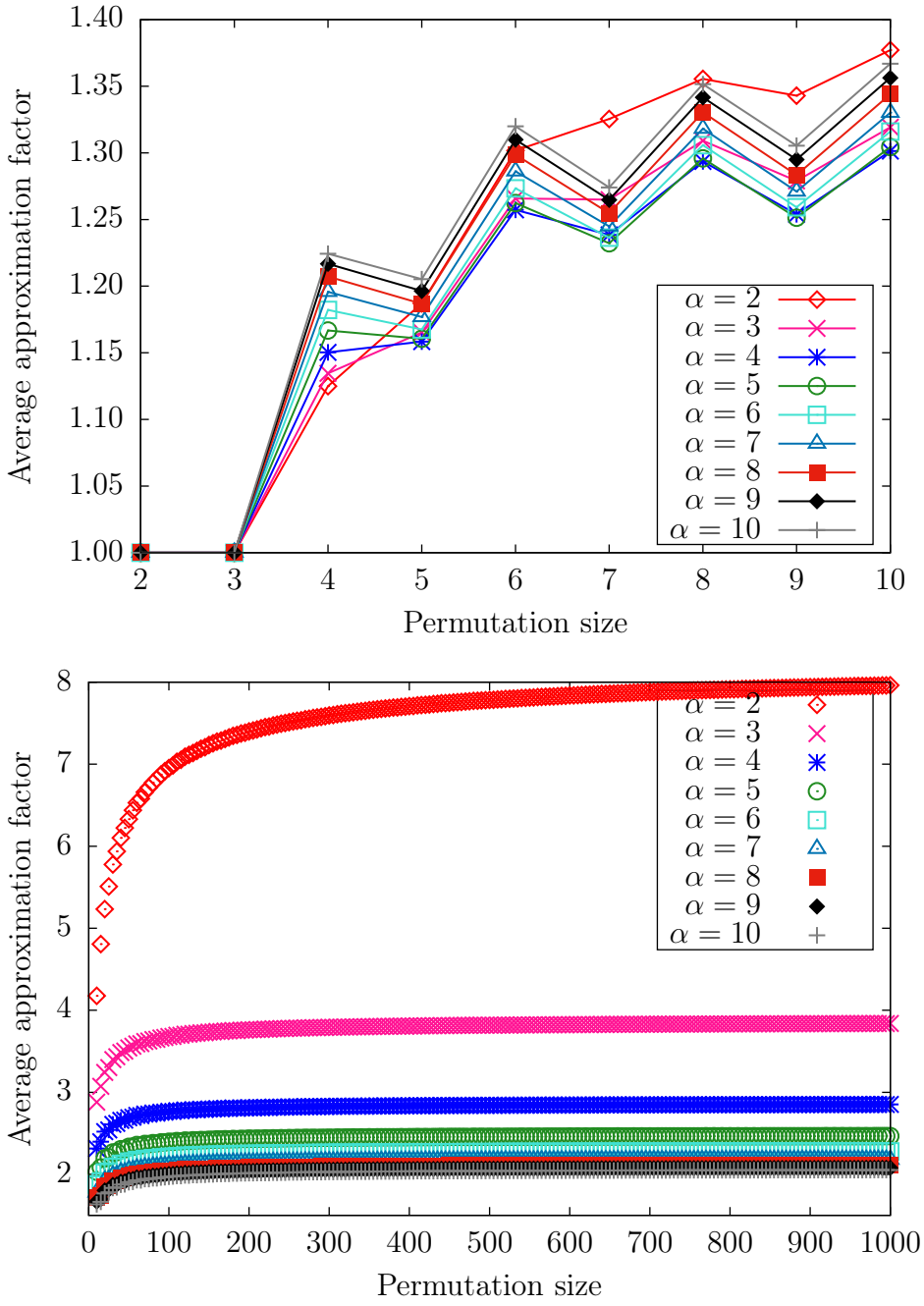


Figure 6.7: Average approximation factors for SBWPR and $\alpha \in \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ while the size of the permutation increases.

Table 6.1: Worst approximation factors for all tested permutations of a given size n for SBWPR when $\alpha \in [2..10]$. The theoretical approximation factor is calculated with the formula $2^\alpha/(2^\alpha - 2) + (2^{2\alpha+1})/(2^\alpha - 2)^2$ given in Equation (6.8).

n	$\alpha = 2$	$\alpha = 3$	$\alpha = 4$	$\alpha = 5$	$\alpha = 6$	$\alpha = 7$	$\alpha = 8$	$\alpha = 9$	$\alpha = 10$
2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
4	1.960	1.879	1.855	1.859	1.875	1.896	1.916	1.934	1.949
5	2.133	1.909	1.855	1.859	1.875	1.896	1.916	1.934	1.949
6	2.538	2.336	2.250	2.213	2.192	2.176	2.160	2.144	2.128
7	2.590	2.336	2.250	2.213	2.192	2.176	2.168	2.159	2.147
8	2.806	2.574	2.466	2.409	2.370	2.337	2.305	2.275	2.247
9	2.865	2.574	2.466	2.409	2.370	2.338	2.319	2.297	2.274
10	5.240	3.591	3.049	2.776	2.620	2.521	2.452	2.398	2.354
15	5.858	3.751	3.030	2.700	2.523	2.415	2.342	2.288	2.245
20	6.182	3.874	3.123	2.905	2.795	2.726	2.677	2.637	2.602
25	6.355	3.805	3.063	2.735	2.561	2.455	2.382	2.328	2.284
30	6.597	3.905	3.077	2.717	2.530	2.418	2.343	2.288	2.245
35	6.632	3.872	3.083	2.750	2.577	2.472	2.400	2.346	2.302
40	6.742	3.913	3.060	2.720	2.544	2.438	2.365	2.311	2.268
45	6.800	3.924	3.073	2.715	2.530	2.418	2.343	2.288	2.245
50	6.918	3.943	3.073	2.704	2.514	2.400	2.325	2.270	2.227
100	7.316	3.955	3.015	2.631	2.437	2.324	2.250	2.199	2.161
150	7.524	3.935	2.999	2.619	2.427	2.315	2.243	2.192	2.155
200	7.626	3.936	2.973	2.586	2.392	2.281	2.211	2.162	2.128
250	7.717	3.924	2.955	2.570	2.381	2.274	2.205	2.158	2.124
300	7.777	3.921	2.942	2.552	2.363	2.255	2.187	2.141	2.109
350	7.798	3.907	2.936	2.550	2.359	2.251	2.183	2.138	2.106
400	7.868	3.908	2.939	2.553	2.361	2.253	2.185	2.139	2.107
450	7.891	3.910	2.934	2.555	2.367	2.259	2.192	2.146	2.113
500	7.904	3.914	2.928	2.543	2.355	2.248	2.182	2.137	2.105
600	7.957	3.914	2.937	2.550	2.358	2.250	2.182	2.137	2.105
700	8.001	3.906	2.921	2.532	2.342	2.234	2.168	2.125	2.094
800	8.005	3.900	2.917	2.530	2.340	2.233	2.167	2.124	2.093
900	8.019	3.898	2.909	2.521	2.330	2.224	2.159	2.116	2.087
1000	8.035	3.900	2.912	2.525	2.335	2.229	2.163	2.120	2.090
Theor. Factor $\leq$	10	4.889	3.756	3.343	3.164	3.080	3.040	3.020	3.010

6.6 Sorting by Length-Weighted Reversals

Bender *et al.* [5] extensively studied sorting permutations and binary strings by length-weighted reversals for $f(\ell) = \ell^\alpha$ when $\alpha > 0$. However, they did not present any approximation algorithm for sorting permutations when $0 < \alpha < 1$.

Our idea here is similar to the one presented in Section 6.3, that is, to adapt algorithm `3-PR`. Therefore, the first step is to note that it is possible to give a lower bound that depends on the number of breakpoints of a permutation, as Lemma 57 shows.

Lemma 57. *For any unsigned permutation π and $\alpha > 0$,*

$$d_r^\alpha(\pi) \geq 2^\alpha \frac{b_{gr}(\pi)}{2}.$$

Proof. First note that one reversal can remove at most two breakpoints. Therefore, to sort a permutation only with reversals at least $b_{gr}(\pi)/2$ such rearrangements will be needed. Since each of these reversals will have length $\ell \geq 2$, the cost of each of them should also be $\ell^\alpha \geq 2^\alpha$. Hence, to sort π the cost is at least $2^\alpha b_{gr}(\pi)/2$. □

Now consider an algorithm that uses the same idea as `3-PR`, but uses only reversals: while the permutation is not sorted, at each step find the highest element that is out of order, reverse the strip that contains this element to turn it into a decreasing strip if necessary, and put this string in the correct position. Such algorithm is showed in Algorithm 26 and it is an $O(n^\alpha)$-approximation for SbWR when $0 < \alpha < 1$, as Theorem 24 shows.

Algorithm 26 Algorithm `3-PR` adapted for SbWR.

3-PR-for-SbWR(π, n)

Input: permutation π and its size n
Output: cost used to sort π

```
1  c ← 0
2  while π ≠ ι_n do
3      Let i be the highest element such that π_i ≠ i
4      Let b be the position of the first element of the strip that contains i
5      Let e be the position of the last element of the strip that contains i
6      if π_b ≠ i then      // If the strip is increasing, transform it into a decreasing one
7          π ← π · ρ(b, e)
8          c ← c + f(e − b + 1)
9      π ← π · ρ(b, i)
10     c ← c + f(i − b + 1)
11 return c
```

Theorem 24. *For $0 < \alpha < 1$,* SbWR *is $O(n^\alpha)$-approximable.*

Proof. In Algorithm 26 we can see that at most two reversals are used to put the strip that contains the highest element in its right position at each iteration. This removes at least one breakpoint. Since each reversal will have a cost of at most n^α, this implies a cost

of at most $2n^\alpha b_{gr}(\pi)$ to sort the permutation π. By the lower bound given in Lemma 57,

$$2n^\alpha b_{gr}(\pi) \leq 2n^\alpha \frac{2d_r^\alpha(\pi)}{2^\alpha} = \frac{4n^\alpha}{2^\alpha} d_r^\alpha(\pi),$$

which means that such algorithm is a $(4/2^\alpha)n^\alpha$-approximation for SbWR. □

6.7 Sorting by Length-Weighted Transpositions and Sorting by Length-Weighted Reversals and Transpositions

In this section we show that all the known results for SbWR and SBbWR can be used for SbWT and SBbWT or for SbWRT and SBbWRT.

First, we present the algorithms given by Bender *et al.* [5] for SBbWR and SbWR. For the former problem, `ZerOneSort_DivideConquer` was given and it is showed in Algorithm 27. For the latter, `PermutationSort_DivideConquer` was given and it is showed in Algorithm 28. Recall the definition of the operator M given in Section 2.2.

Algorithm 27 Sorting algorithm for SBbWR.

ZerOneSort_DivideConquer(T, n)
Input: binary string T and its size n
Output: cost used to sort T
1 **if** T is sorted **then**
2 **return** 0
3 $x \leftarrow \lfloor n/2 \rfloor$
4 $c_1 \leftarrow$ ZerOneSort_DivideConquer($t_1 t_2 \dots t_x,\ x$)
5 $c_2 \leftarrow$ ZerOneSort_DivideConquer($t_{x+1} t_{x+2} \dots t_n,\ n-x$)
// Now T is of the form $0^{w_1}1^{w_2}0^{w_3}1^{w_4}$
6 Let i be the smallest position such that $\pi_i = 1$
7 Let j be the highest position such that $\pi_j = 0$
8 $T \leftarrow T \cdot \rho(i, j)$
9 **return** $c_1 + c_2 + f(j - i + 1)$

We can see that these algorithms can be adapted straightforwardly when transpositions are allowed: note that the reversal on `ZerOneSort_DivideConquer` results in exchanging a block of 1's with a block of 0's, which can be easily mimicked by a transposition that has the same length. This means that the upper bounds for the diameter of sorting binary strings and permutations by reversals that are given by such algorithms for $0 < \alpha < 2$ are the same for sorting when transpositions are allowed. This leads to Lemma 58.

Lemma 58. *For* $\beta \in \{t, rt\}$ *and* $0 < \alpha < 2$,

$$C_\beta^\alpha(n) \text{ is } \begin{cases} O(n) & \text{if } 0 < \alpha < 1 \\ O(n \lg n) & \text{if } \alpha = 1 \\ O(n^\alpha) & \text{if } 1 < \alpha < 2 \end{cases} \tag{6.9}$$

Algorithm 28 Sorting algorithm for SBWR.

PERMUTATIONSORT_DIVIDECONQUER(π, n)

Input: permutation π and its size n
Output: cost used to sort π

```
1  if n ≤ 1 or π is sorted then
2      return 0
3  T ← M(n, ⌈n/2⌉, π) // Section 2.2
4  c ← ZEROONESORT_DIVIDECONQUER(T, n)
5  Apply reversals that sorted T on π
6  x ← ⌈n/2⌉
7  c1 ← PERMUTATIONSORT_DIVIDECONQUER((π1, π2, ..., πx), x)
8  c2 ← PERMUTATIONSORT_DIVIDECONQUER((πx+1, πx+2, ..., πn), n − x)
9  return c + c1 + c2
```

and

$$D^{\alpha}_{\beta}(n) \text{ is } \begin{cases} O(n \lg n) & \text{if } 0 < \alpha < 1 \\ O(n \lg^2 n) & \text{if } \alpha = 1 \\ O(n^{\alpha}) & \text{if } 1 < \alpha < 2. \end{cases} \tag{6.10}$$

In fact, except for the approximation algorithms for $\alpha = 1$, all other results given by Bender *et al.* [5] regarding approximation algorithms and bounds for the diameter can be quite easily adapted for the cases when transpositions are allowed. For $\alpha = 1$, the approximation algorithms given by Pinter and Skiena [50] can be used.

Lemmas 46 and 57 showed lower bounds on the number of breakpoints for the other problems, which also holds when transpositions are allowed, as Lemma 59 shows. Therefore, for $0 < \alpha < 1$, the $O(n^{\alpha})$-approximation algorithms we presented in Sections 6.3 and 6.6 can also be adapted for sorting when transpositions are allowed.

Lemma 59. *For any unsigned permutation π and $\alpha > 0$,*

$$d^{\alpha}_{t}(\pi) \geq 2^{\alpha}\frac{b_g(\pi)}{3} \text{ and } d^{\alpha}_{rt}(\pi) \geq 2^{\alpha}\frac{b_{gr}(\pi)}{3}.$$

Proof. First note that one transposition can remove at most three breakpoints. Hence, to sort a permutation only with transpositions at least $b_g(\pi)/3$ such rearrangements will be needed. Since each of these transpositions will have length $\ell \geq 2$, the cost of each of them should also be $\ell^{\alpha} \geq 2^{\alpha}$. Therefore, to sort π the cost is at least $2^{\alpha}b_g(\pi)/3$.

When both reversals and transpositions are allowed, at least $b_{gr}(\pi)/3$ rearrangements are needed, each one with cost at least 2^{α}, which means that the cost to sort π is also at least $2^{\alpha}b_g(\pi)/3$. □

These adaptations guarantee the results in Theorem 25.

Theorem 25. *For $\beta \in \{t, rt\}$:*

1. $C^{\alpha}_{\beta}(n)$ *is* $\begin{cases} \Theta(n) & \text{if } 0 < \alpha < 1 \\ \Theta(n \lg n) & \text{if } \alpha = 1 \\ \Theta(n^{\alpha}) & \text{if } 1 < \alpha < 2 \\ \Theta(n^2) & \text{if } \alpha \geq 2. \end{cases}$

2. $D^{\alpha}_{\beta}(n)$ *is* $\begin{cases} \Omega(n) \text{ and } O(n \lg n) & \text{if } 0 < \alpha < 1 \\ \Omega(n \lg n) \text{ and } O(n \lg^2 n) & \text{if } \alpha = 1 \\ \Theta(n^{\alpha}) & \text{if } 1 < \alpha < 2 \\ \Theta(n^2) & \text{if } \alpha \geq 2. \end{cases}$

3. *For* $0 < \alpha < 1$, *there is a* $O(1)$*-approximation algorithm for* SBBWT *and* SBBWRT *and there is a* $O(n^{\alpha})$*-approximation for* SBWT *and* SBWRT*;*

4. *For* $\alpha = 1$, *there is a* $O(\lg n)$*-approximation for* SBBWT *and* SBBWRT*, and there is a* $O(\lg^2 n)$*-approximation algorithm for* SBWT *and* SBWRT*;*

5. *For* $1 < \alpha < 2$, *there is a* $O(1)$*-approximation algorithm for* SBBWT *and* SBBWRT*, and there is a* $O(\lg n)$*-approximation algorithm for* SBWT *and* SBWRT*;*

6. *For* $\alpha \geq 2$, SBBWT *and* SBBWRT *are polynomial and there is a 2-approximation for* SBWT *and* SBWRT*;*

7. *When* $\alpha \geq 3$, SBWT *and* SBWRT *are polynomially solvable.*

6.8 Sorting Signed Permutations and Signed Binary Strings by Length-Weighted Prefix and Suffix Rearrangements

In order to use the algorithms already developed in the previous sections over signed permutations, we will use the concept of image of a signed permutation [37].

Definition 24. *The* image *of a signed permutation* π *is defined as the unsigned permutation* $\pi' = (\pi'_1 \ \ldots \ \pi'_{2n})$*, where* $\pi'_{2i-1} = 2\pi_i - 1$ *and* $\pi'_{2i} = 2\pi_i$ *if* $\pi_i > 0$ *or* $\pi'_{2i-1} = -2\pi_i$ *and* $\pi'_{2i} = -2\pi_i - 1$ *if* $\pi_i < 0$*.*

If π' is the image of a signed permutation π, then it is easy to see that any sorting sequence for π' can be mimicked over π if we never separate elements π'_{2i-1} and π'_{2i}.

Example 24. *Let* $\pi = (4\ 6\ {-3}\ {-5}\ 2\ {-1})$*. So, the image of* π *is* $\pi' = (7\ 8\ 11\ 12\ 6\ 5\ 10\ 9\ 3\ 4\ 2\ 1)$*. Note that the sequence* $\rho_p(4)$, $\rho_p(12)$, $\rho_p(4)$, $\rho_p(2)$, $\rho_p(6)$, $\rho_p(10)$, *and* $\rho_p(8)$ *sorts* π'*. Also, all prefix reversals are applied over even positions, which means it does not separate elements* π'_{2i-1} *and* π'_{2i}*. Based on this sequence, we create* $\bar{\rho}_p(2)$, $\bar{\rho}_p(6)$, $\bar{\rho}_p(2)$, $\bar{\rho}_p(1)$, $\bar{\rho}_p(3)$, $\bar{\rho}_p(5)$, *and* $\bar{\rho}_p(4)$ *by dividing by 2 each position of the previous sequence. It is easy to see that this new sequence sorts* π*.*

With the concept of image, we can use the algorithms presented in Section 6.1 to create algorithms for SBWP$\bar{\text{R}}$, SBBWP$\bar{\text{R}}$, SBWPS$\bar{\text{R}}$, SBBWPS$\bar{\text{R}}$, SBWP$\bar{\text{R}}$T, SBBWP$\bar{\text{R}}$T, SBWPS$\bar{\text{R}}$T, and SBBWPS$\bar{\text{R}}$T when $\alpha = 1$. The only difference is that if the median of the interval is not an even number, we increase it by one unit, so that the algorithms can act over intervals that also exist in the signed permutation. Also, the lengths of the rearrangements must always be divided by 2 before we apply the cost function. It is easy to see that these slightly-changed algorithms are still $O(\lg^2 n)$-approximation algorithms when $\alpha = 1$ for signed permutations.

Example 25. *The following example shows the execution of the adaptation of* WPR *over* $\pi' = (7\ 8\ 11\ 12\ 6\ 5\ 10\ 9\ 3\ 4\ 2\ 1)$*, which is the image of* $\pi = (4\ 6\ {-3}\ {-5}\ 2\ {-1})$ *given in the previous example. We will refer to such adaptation as* $\texttt{WP}\bar{\texttt{R}}$ *and the adaptation of the partition algorithm as* $\texttt{partitionWP}\bar{\texttt{R}}$*. We will say that the first call, which is* $\texttt{WP}\bar{\texttt{R}}(\pi,12,\text{INC})$*, is call (0).*

$\pi' = (7\ 8\ 11\ 12\ 6\ 5\ 10\ 9\ 3\ 4\ 2\ 1)$ // *no base case applies, so calls partition*
$\pi' \leftarrow \texttt{partitionWP}\bar{\texttt{R}}(\pi',12,\text{DEC},6) = (9\ 10\ 7\ 8\ 11\ 12\ 6\ 5\ 3\ 4\ 2\ 1)$ // *(1) calls* $\texttt{WP}\bar{\texttt{R}}(\pi',6,\text{DEC})$*; no base case*
$\pi' \leftarrow \texttt{partitionWP}\bar{\texttt{R}}(\pi',6,\text{INC},10) = (9\ 10\ 7\ 8\ 11\ 12\ 6\ 5\ 3\ 4\ 2\ 1)$ // *(2) calls* $\texttt{WP}\bar{\texttt{R}}(\pi',4,\text{INC})$*; no base case*
$\pi' \leftarrow \texttt{partitionWP}\bar{\texttt{R}}(\pi',4,\text{DEC},8) = (\underline{9\ 10}\ 7\ 8\ 11\ 12\ 6\ 5\ 3\ 4\ 2\ 1)$ // *(3) calls* $\texttt{WP}\bar{\texttt{R}}(\pi',2,\text{DEC})$*; has base case*
$\pi' \leftarrow \pi' \cdot \rho_p(2) = (\underline{10\ 9\ 7\ 8}\ 11\ 12\ 6\ 5\ 3\ 4\ 2\ 1)$ // *returns to (2); reverse whole interval*
$\pi' \leftarrow \pi' \cdot \rho_p(4) = (\underline{8\ 7}\ 9\ 10\ 11\ 12\ 6\ 5\ 3\ 4\ 2\ 1)$ // *(4) calls* $\texttt{WP}\bar{\texttt{R}}(\pi',2,\text{INC})$*; has base case*
$\pi' \leftarrow \pi' \cdot \rho_p(2) = (\underline{7\ 8\ 9\ 10\ 11\ 12}\ 6\ 5\ 3\ 4\ 2\ 1)$ // *returns to (2); returns to (1); reverse interval*
$\pi' \leftarrow \pi' \cdot \rho_p(6) = (12\ 11\ 10\ 9\ 8\ 7\ 6\ 5\ 3\ 4\ 2\ 1)$ // *(5) calls* $\texttt{WP}\bar{\texttt{R}}(\pi',2,\text{DEC})$*; is DEC sorted*
$\pi' = (\underline{12\ 11\ 10\ 9\ 8\ 7\ 6\ 5\ 3\ 4\ 2\ 1})$ // *returns to (0); reverse whole interval*
$\pi' \leftarrow \pi' \cdot \rho_p(12) = (1\ 2\ 4\ 3\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12)$ // *(6) calls* $\texttt{WP}\bar{\texttt{R}}(\pi',6,\text{INC})$*; no base case*
$\pi' \leftarrow \texttt{partitionWP}\bar{\texttt{R}}(\pi',6,\text{DEC},4) = (6\ 5\ 3\ 4\ 2\ 1\ 7\ 8\ 9\ 10\ 11\ 12)$ // *(7) calls* $\texttt{WP}\bar{\texttt{R}}(\pi',2,\text{DEC})$*; is DEC sorted*
$\pi' = (\underline{6\ 5\ 3\ 4\ 2\ 1}\ 7\ 8\ 9\ 10\ 11\ 12)$ // *returns to (6); reverse whole interval*
$\pi' \leftarrow \pi' \cdot \rho_p(6) = (1\ 2\ 4\ 3\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12)$ // *(8) calls* $\texttt{WP}\bar{\texttt{R}}(\pi',4,\text{INC})$*; no base case*
$\pi' \leftarrow \texttt{partitionWP}\bar{\texttt{R}}(\pi',4,\text{DEC},2) = (\underline{3\ 4}\ 2\ 1\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12)$ // *(9) calls* $\texttt{WP}\bar{\texttt{R}}(\pi',2,\text{DEC})$*; has base case*
$\pi' \leftarrow \pi' \cdot \rho_p(2) = (\underline{4\ 3\ 2\ 1}\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12)$ // *returns to (8); reverse whole interval*
$\pi' \leftarrow \pi' \cdot \rho_p(4) = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12)$ // *(10) calls* $\texttt{WP}\bar{\texttt{R}}(\pi',2,\text{INC})$*; is INC sorted*
$\pi' = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12)$ // *returns to (8); returns to (0)*

Note that calls (7) and (8) should actually be $\texttt{WP}\bar{\texttt{R}}(\pi', 3, INC/DEC)$*, since we want to sort an interval with 6 elements, but this would create rearrangements that are invalid over the signed permutation. Therefore, the algorithm considers the median of the interval* $[1..6]$ *in permutation* $(1\ 2\ 4\ 3\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12)$ *as* 4*. The partition call over this permutation generates* $(6\ 5\ 3\ 4\ 2\ 1\ 7\ 8\ 9\ 10\ 11\ 12)$*, the first recursive call (7) to* $\texttt{WP}\bar{\texttt{R}}$ *is over interval* $[1..2]$ *(elements greater than the median) and it does not change the permutation, and the second recursive call (8) to* $\texttt{WP}\bar{\texttt{R}}$ *is over interval* $[1..4]$ *of permutation* $(1\ 2\ 4\ 3\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12)$ *(elements less than or equal to the median).*

The use of those algorithms means that the upper bound on the diameters given in Lemma 56 are also valid for the problems that we are considering in this section.

Also note that, if π is a signed permutation and π^u is π without the signs (i.e, π^u is unsigned), then $d^\alpha_\beta(\pi) \geq d^\alpha_\beta(\pi^u)$, which means that every lower bound that we found for the unsigned case is also valid for the signed case. This is also true for signed binary strings.

It is also fairly easy to see that the approximation algorithms given in Sections 6.3 and 6.4 can be adapted for signed problems that consider prefix rearrangements if we take care of not separating elements π'_{2i-1} and π'_{2i} and divide the lengths of the rearrangements before applying the cost function.

These adaptations guarantee the results in Theorems 26 and 27.

Theorem 26. *For* $\beta \in \{p\bar{r}, p\bar{r}t\}$*:*

1. *For* $0 < \alpha < 1$*,* $C^\alpha_\beta(n)$ *is* $\Theta(n)$ *while* $D^\alpha_\beta(n)$ *is* $\Omega(n)$ *and* $O(n \lg n)$*, there is a* $O(\lg n)$*-approximation algorithm for sorting binary strings, and there is a* $O(n^\alpha)$*-approximation for sorting permutations;*

2. *For* $\alpha = 1$*,* $C^\alpha_\beta(n)$ *is* $\Theta(n \lg n)$ *while* $D^\alpha_\beta(n)$ *is in* $\Omega(n \lg n)$ *and* $O(n \lg^2 n)$*, there is a* $O(\lg n)$*-approximation algorithm for sorting binary strings, and there is a* $O(\lg^2 n)$*-approximations for sorting permutations;*

3. *For $\alpha > 1$, both $C^{\alpha}_{\beta}(n)$ and $D^{\alpha}_{\beta}(n)$ are $\Theta(n^{\alpha})$ and there are $O(1)$-approximation algorithms for sorting both binary strings and permutations.*

Theorem 27. *For $\beta \in \{ps\bar{r}, ps\bar{r}t\}$, there exists a $O(\lg n)$-approximation algorithm for sorting binary strings and there exists a $O(\lg^2 n)$-approximation for sorting permutations.*

6.9 Summary of the Chapter

Table 6.2 summarizes the best approximation factors and bounds for the diameters of the problems that were studied in this chapter. Note that the results for sorting by prefix and suffix rearrangements are the approximation algorithms with factor $O(\lg n)$ for binary strings and $O(\lg^2 n)$ for permutations and they are valid only for $\alpha = 1$. There are no results for other values of α because we were not able to find good lower bounds on the distance for such problems yet.

Table 6.2: Summary of the results obtained for length-weighted rearrangement problems.

Rearrangements	α	**Approximation Factor**		**Diameter**		
		Bin. Str.	**Perm.**	**Bin. Str.**	**Perm. (Lower Bound)**	**Perm. (Upper Bound)**
Pref. Reversals, Pref. Transpositions, and Pref. Reversals and Transpositions	$0 < \alpha < 1$	$O(\lg n)$ (Thm. 19)	$O(n^\alpha)$ (Thms. 20 and 21)	$\Theta(n)$ (Thm. 17)	$\Omega(n)$ (Lems. 48, 49, and 50)	$O(n \lg n)$ (Lem. 56)
	$\alpha = 1$	$O(\lg n)$ (Thms. 8, 13, and 16)	$O(\lg^2 n)$ (Thms. 7, 11, and 14)	$\Theta(n \lg n)$ (Thm. 17)	$\Omega(n \lg n)$ (Lems. 51 and 54)	$O(n \lg^2 n)$ (Lem. 56)
	$1 < \alpha < 2$	$O(1)$ (Thm. 22)	$O(1)$ (Thm. 22)	$\Theta(n^\alpha)$ (Thm. 17)	$\Theta(n^\alpha)$ (Thm. 18)	
	$2 \leq \alpha < 3$	4 (Cor. 23)	10 (Cor. 23)			
	$\alpha \geq 3$	3 (Cor. 23)	5 (Cor. 23)			
	$\alpha \to \infty$	$(2^{\alpha+1})/(2^\alpha - 2)$ (Cor. 23)	$2^\alpha/(2^\alpha - 2) + (2^{2\alpha+1})/(2^\alpha - 2)^2$ (Cor. 23)			
Pref. and Suf. Reversals, Pref. and Suf. Transpositions, and Pref. and Suf. Reversals and Transpositions	$\alpha = 1$	$O(\lg n)$ (Thms. 10, 13 and 16)	$O(\lg^2 n)$ (Thms. 9, 12, and 15)	-	-	-
Reversals	$0 < \alpha < 1$	-	$O(n^\alpha)$	-	-	-
Transpositions, Reversals and Transpositions (Sec. 6.7)	$0 < \alpha < 1$	$O(1)$	$O(n^\alpha)$	$\Theta(n)$	$\Omega(n)$	$O(n \lg n)$
	$\alpha = 1$	$O(\lg n)$	$O(\lg^2 n)$	$\Theta(n \lg n)$	$\Omega(n \lg n)$	$O(n \lg^2 n)$
	$1 < \alpha < 2$	$O(1)$	$O(\lg n)$	$\Theta(n^\alpha)$	$\Theta(n^\alpha)$	
	$2 \leq \alpha < 3$	1	2	$\Theta(n^2)$	$\Theta(n^2)$	
	$\alpha \geq 3$		1			
Sig. Pref. Reversals and Sig. Pref. Reversals and Transpositions (Sec. 6.8)	$0 < \alpha < 1$	$O(\lg n)$	$O(n^\alpha)$	$\Theta(n)$	$\Omega(n)$	$O(n \lg n)$
	$\alpha = 1$	$O(\lg n)$	$O(\lg^2 n)$	$\Theta(n \lg n)$	$\Omega(n \lg n)$	$O(n \lg^2 n)$
	$\alpha > 1$	$O(1)$	$O(1)$	$\Theta(n^\alpha)$	$\Theta(n^\alpha)$	
Sig. Pref. and Suf. Reversals and Sig Pref. and Suf. Reversals and Transpositions (Sec. 6.8)	$\alpha = 1$	$O(\lg n)$	$O(\lg^2 n)$	-	-	-

Chapter 7

Results Obtained for Exponential Cost Function

We acknowledge that the usual cost function when length-weighted rearrangements are considered is the polynomial cost function that was used so far. In this chapter, however, we present results for an exponential cost function $f(\ell) = 2^\ell$. Unless specified otherwise, all problems mentioned in this chapter consider such cost function. Our motivation in doing so is based on the studies that exist over short and super short rearrangements [32, 33]. Such types of rearrangements are restricted to a very small portion of the permutation and, due to this cost function, using bigger rearrangements is not so preferred either.

For this new cost function, the only adaptation on the definitions that were given so far is related to some notation, as given next.

Definition 25. *Given a rearrangement model β, the distance to sort a permutation π is denoted as $d^e_\beta(\pi)$, the distance to sort a binary string T is $c^e_\beta(T)$, the diameter for sorting permutations is denoted as $D^e_\beta(n)$, and the diameter for sorting binary strings is $C^e_\beta(n)$.*

In Section 7.1 we present results for sorting binary strings and permutations when prefix rearrangements are being considered. In Section 7.2 we present results for SbWR and SBbWR. In Section 7.3 we present results for SbWT, SbWRT, SBbWT, and SBbWRT. In Section 7.4 we present results for the signed variants of sorting by prefix rearrangements. Lastly, in Section 7.5 we give a summary of all results presented in this chapter.

7.1 Sorting by Length-Weighted Prefix Reversals, Sorting by Length-Weighted Prefix Transpositions, and Sorting by Length-Weighted Prefix Reversals and Transpositions

We start by giving a lower bound on the distance in Lemma 60.

Lemma 60. *Let $\beta \in \{pr, pt, prt\}$. For any permutation π and binary string T with n valid elements,*

$$c^e_\beta(T) \geq 2^n \text{ and } d^e_\beta(\pi) \geq 2^n.$$

Proof. For binary strings, since $t_n \neq 1$ by Definition 21, at some point one rearrangement of length n will have to be performed in order to sort the binary string, which will cost at least $f(n) = 2^n$.

Likewise, for permutations, at least one rearrangement that places the element n in its right position should be done, which will also cost $f(n) = 2^n$. □

Consider Algorithm 29, which sorts a binary string by always placing the rightmost 1 which is not in the last block of 1's next to such block. Such algorithm is a 3-approximation for SBBWPR, as Theorem 28 shows.

Algorithm 29 A 3-approximation algorithm for SBBWPR.

3-BWPR(T, n)

Input: binary string T and its size n
Output: cost used to sort T

$c \leftarrow 0$
while $\pi \neq \iota_n$ **do**
  Let y be the highest position such that $t_y = 0$
  Let $x < y$ be the highest position such that $t_x = 1$ and $t_i = 0$ for all $x + 1 \leq i \leq y$
  if x is not the last position of the first block **then**
    $T \leftarrow T \cdot \rho_p(x)$
    $c \leftarrow c + f(x)$
  $T \leftarrow T \cdot \rho_p(y)$
  $c \leftarrow c + f(y)$
return c

Theorem 28. *For $f(\ell) = 2^\ell$,* SBBWPR *is 3-approximable.*

Proof. Suppose that $T = t_1 t_2 \dots t_n$ has k 0's and, therefore, $n - k$ 1's. Note that Algorithm 29 sorts T by performing at most $n - k$ iterations and that each iteration contains at most two prefix reversals. The maximum cost is reached when it performs the prefix reversals with the largest length possible. In a given iteration, the algorithm places at least one 1 at a certain position y, which is defined as the highest position before the last block of 1's; for the second prefix reversal of such iteration to have the largest length possible, such 1 must be in position $y - 1$ at the beginning of the iteration. Therefore, this algorithm will sort a bit sequence T with a cost of at most

$$\sum_{i=k+1}^{n} (f(i-1) + f(i)) = \sum_{i=k+1}^{n} (2^{i-1} + 2^i) = 3 \times 2^n - 3 \times 2^k \leq 3 \times 2^n,$$

which means that, according to the lower bound given in Lemma 60, this is an approximation algorithm of factor 3 for SBBWPR. □

Example 26. *The following example shows the execution of Algorithm 29 over $T =$*

011000110111010*:*

$$
\begin{aligned}
T &= \underline{011000110111010}\ \text{// } y = 15 \text{ and } x = 14\text{; bring rightmost 1 to beginning}\\
T \leftarrow T{\cdot}\rho_p(14) &= \underline{101110110001100}\ \text{// put first 1 in the end}\\
T \leftarrow T{\cdot}\rho_p(15) &= \underline{001100011011101}\ \text{// } y = 14 \text{ and } x = 13\text{; bring rightmost 1 (not in the last block) to beginning}\\
T \leftarrow T{\cdot}\rho_p(13) &= \underline{111011000110001}\ \text{// put first 1 in the end (before last block)}\\
T \leftarrow T{\cdot}\rho_p(14) &= \underline{000110001101111}\ \text{// } y = 11 \text{ and } x = 10\\
T \leftarrow T{\cdot}\rho_p(10) &= \underline{110001100001111}\ \text{// put first 1 in the end}\\
T \leftarrow T{\cdot}\rho_p(14) &= \underline{000011000111111}\ \text{// } y = 9 \text{ and } x = 6\\
T \leftarrow T{\cdot}\rho_p(6) &= \underline{110000000111111}\ \text{// put first 1 in the end}\\
T \leftarrow T{\cdot}\rho_p(9) &= 000000011111111
\end{aligned}
$$

When prefix transpositions are allowed, it suffices for the algorithm to perform only one prefix transposition to place the rightmost 1 which is not in the last block of 1's near to such block, as Algorithm 30 shows. This guarantees an approximation factor of 2 for SBBWPT and for SBBWPRT, as Theorem 29 shows.

Algorithm 30 A 2-approximation algorithm for both SBBWPT and SBBWPRT.

2-BWPT/2-BWPRT(T, n)

Input: binary string T and its size n
Output: cost used to sort T

```
1  c ← 0
2  while π ≠ ι_n do
3      Let y be the highest position such that t_y = 0
4      Let x < y be highest the position such that t_x = 1 and t_i = 0 for all x + 1 ≤ i ≤ y
5      T ← T · τ_p(x + 1, y + 1)
6      c ← c + f(y)
7  return c
```

Theorem 29. *For $f(\ell) = 2^\ell$,* SBBWPT *and* SBBWPRT *are 2-approximable.*

Proof. The adapted algorithm given in Algorithm 30 sorts a bit sequence T that has k 0's and $n-k$ 1's with a cost of at most

$$\sum_{i=k+1}^{n} f(i) = \sum_{i=k+1}^{n} 2^i = 2 \times 2^n - 2 \times 2^k \leq 2 \times 2^n.$$

Therefore, according do the lower bound given in Lemma 60, this is a 2-approximation algorithm for SBBWPT and for SBBWPRT. □

A similar approach can be used for sorting permutations: at each step, bring the highest element that is out of order to the beginning of the permutation and put this element in its correct position (see Algorithm 31). This is in fact a 4-approximation algorithm for SBPR. Also, it is a 3-approximation for SBWPR, as Theorem 30 shows.

Theorem 30. *For $f(\ell) = 2^\ell$,* SBWPR *is 3-approximable.*

Proof. In Algorithm 31, note that if element i is the highest out of order, then its position is at most $i-1$. Therefore, considering length-weighted prefix reversals, this algorithm will sort a permutation π with a cost of at most

$$\sum_{i=2}^{n}(f(i-1) + f(i)) = \sum_{i=2}^{n}(2^{i-1} + 2^i) = 3 \times 2^n - 6 \leq 3 \times 2^n.$$

Algorithm 31 A 3-approximation algorithm for SbWPR.

3-WPR(π, n)

Input: permutation π and its size n
Output: cost used to sort π

$c \leftarrow 0$
while $\pi \neq \iota_n$ **do**
  Let $\pi_j = i$ be the highest element such that $\pi_i \neq i$
  if $j \neq 1$ **then** // Bring the element to the beginning if necessary
    $\pi \leftarrow \pi \cdot \rho_p(j)$
    $c \leftarrow c + f(j)$
  // Move i to its right position
  $\pi \leftarrow \pi \cdot \rho_p(i)$
  $c \leftarrow c + f(i)$
return d

Therefore, according to the lower bound given in Lemma 60, it is a 3-approximation algorithm. □

Example 27. *The following example shows the execution of Algorithm 31 over* $\pi = (9\ 6\ 2\ 14\ 10\ 15\ 7\ 12\ 4\ 11\ 3\ 8\ 13\ 1\ 5)$*:*

$\pi = (\underline{9\ 6\ 2\ 14\ 10\ 15}\ 7\ 12\ 4\ 11\ 3\ 8\ 13\ 1\ 5)$ // $\pi_6 = 15$ *is the highest out of place; bring it to beginning*
$\pi \leftarrow \pi \cdot \rho_p(6) = (\underline{15\ 10\ 14\ 2\ 6\ 9\ 7\ 12\ 4\ 11\ 3\ 8\ 13\ 1\ 5})$ // *put it in its right position*
$\pi \leftarrow \pi \cdot \rho_p(15) = (\underline{5\ 1\ 13\ 8\ 3\ 11\ 4\ 12\ 7\ 9\ 6\ 2\ 14}\ 10\ 15)$ // $\pi_{13} = 14$ *is the highest out of place*
$\pi \leftarrow \pi \cdot \rho_p(13) = (\underline{14\ 2\ 6\ 9\ 7\ 12\ 4\ 11\ 3\ 8\ 13\ 1\ 5\ 10}\ 15)$ // *put it in its right position*
$\pi \leftarrow \pi \cdot \rho_p(14) = (\underline{10\ 5\ 1\ 13}\ 8\ 3\ 11\ 4\ 12\ 7\ 9\ 6\ 2\ 14\ 15)$ // $\pi_4 = 13$ *is the highest*
$\pi \leftarrow \pi \cdot \rho_p(4) = (\underline{13\ 1\ 5\ 10\ 8\ 3\ 11\ 4\ 12\ 7\ 9\ 6\ 2}\ 14\ 15)$ // *put it in its right position*
$\pi \leftarrow \pi \cdot \rho_p(13) = (\underline{2\ 6\ 9\ 7\ 12}\ 4\ 11\ 3\ 8\ 10\ 5\ 1\ 13\ 14\ 15)$ // $\pi_5 = 12$ *is the highest*
$\pi \leftarrow \pi \cdot \rho_p(5) = (\underline{12\ 7\ 9\ 6\ 2\ 4\ 11\ 3\ 8\ 10\ 5\ 1}\ 13\ 14\ 15)$ // *put it in its right position*
$\pi \leftarrow \pi \cdot \rho_p(12) = (\underline{1\ 5\ 10\ 8\ 3\ 11}\ 4\ 2\ 6\ 9\ 7\ 12\ 13\ 14\ 15)$ // $\pi_6 = 11$ *is the highest*
$\pi \leftarrow \pi \cdot \rho_p(6) = (\underline{11\ 3\ 8\ 10\ 5\ 1\ 4\ 2\ 6\ 9\ 7}\ 12\ 13\ 14\ 15)$ // *put it in its right position*
$\pi \leftarrow \pi \cdot \rho_p(11) = (\underline{7\ 9\ 6\ 2\ 4\ 1\ 5\ 10}\ 8\ 3\ 11\ 12\ 13\ 14\ 15)$ // $\pi_8 = 10$ *is the highest*
$\pi \leftarrow \pi \cdot \rho_p(8) = (\underline{10\ 5\ 1\ 4\ 2\ 6\ 9\ 7\ 8\ 3}\ 11\ 12\ 13\ 14\ 15)$ // *put it in its right position*
$\pi \leftarrow \pi \cdot \rho_p(10) = (\underline{3\ 8\ 7\ 9}\ 6\ 2\ 4\ 1\ 5\ 10\ 11\ 12\ 13\ 14\ 15)$ // $\pi_4 = 9$ *is the highest*
$\pi \leftarrow \pi \cdot \rho_p(4) = (\underline{9\ 7\ 8\ 3\ 6\ 2\ 4\ 1\ 5}\ 10\ 11\ 12\ 13\ 14\ 15)$ // *put it in its right position*
$\pi \leftarrow \pi \cdot \rho_p(9) = (\underline{5\ 1\ 4\ 2\ 6\ 3\ 8}\ 7\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$ // $\pi_7 = 8$ *is the highest*
$\pi \leftarrow \pi \cdot \rho_p(7) = (\underline{8\ 3\ 6\ 2\ 4\ 1\ 5\ 7}\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$ // *put it in its right position*
$\pi \leftarrow \pi \cdot \rho_p(8) = (\underline{7\ 5\ 1\ 4\ 2\ 6\ 3}\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$ // $\pi_1 = 7$ *is the highest; already in the beginning*
$\pi \leftarrow \pi \cdot \rho_p(7) = (\underline{3\ 6}\ 2\ 4\ 1\ 5\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$ // $\pi_2 = 6$ *is the highest*
$\pi \leftarrow \pi \cdot \rho_p(2) = (\underline{6\ 3\ 2\ 4\ 1\ 5}\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$ // *put it in its right position*
$\pi \leftarrow \pi \cdot \rho_p(6) = (\underline{5\ 1\ 4\ 2\ 3}\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$ // $\pi_1 = 5$ *is the highest; already in the beginning*
$\pi \leftarrow \pi \cdot \rho_p(5) = (\underline{3\ 2\ 4}\ 1\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$ // $\pi_3 = 4$ *is the highest*
$\pi \leftarrow \pi \cdot \rho_p(3) = (\underline{4\ 2\ 3\ 1}\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$ // *put it in its right position*
$\pi \leftarrow \pi \cdot \rho_p(4) = (\underline{1\ 3}\ 2\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$ // $\pi_2 = 3$ *is the highest*
$\pi \leftarrow \pi \cdot \rho_p(2) = (\underline{3\ 1\ 2}\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$ // *put it in its right position*
$\pi \leftarrow \pi \cdot \rho_p(3) = (\underline{2\ 1}\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$ // $\pi_1 = 2$ *is the highest; already in the beginning*
$\pi \leftarrow \pi \cdot \rho_p(2) = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15)$

Again, if we adapt this last algorithm to use prefix transpositions, then only one operation is needed to put the highest element in its correct position (see Algorithm 32). Theorem 31 shows that this is a 2-approximation algorithm for SbWPT and SbWPRT.

Theorem 31. *For* $f(\ell) = 2^\ell$*,* SbWPT *and* SbWPRT *are* 2*-approximable.*

Proof. Since only one operation is needed to put the highest element in its correct place at each iteration, the algorithm will sort a permutation π with a cost of at most

$$\sum_{i=2}^{n} f(i) = \sum_{i=2}^{n} 2^i = 2 \times 2^n - 4 \leq 2 \times 2^n,$$

Algorithm 32 A 2-approximation algorithm for both SBWPT and SBWPRT.

2-WPT/2-WPRT(π, n)

Input: permutation π and its size n
Output: cost used to sort π

```
1  c ← 0
2  while π ≠ ι_n do
3      Let π_j = i be the highest element such that π_i ≠ i
4      π ← π · τ_p(j + 1, i + 1)
5      c ← c + f(i)
6  return c
```

which means that it is, according do the lower bound given in Lemma 60, an approximation algorithm of factor 2 for SBWPT and for SBWPRT. □

Note that these algorithms given so far show that $C^e_\beta(n)$ and $D^e_\beta(n)$ are both $O(2^n)$, for $\beta \in \{pr, pt, prt\}$. The lower bounds given in Lemma 60, on the other hand, show that $C^e_\beta(n)$ and $D^e_\beta(n)$ are both $\Omega(2^n)$. This leads us directly to the following theorem.

Theorem 32. *For $\beta \in \{pr, pt, prt\}$ and $f(\ell) = 2^\ell$, $C^e_\beta(n)$ and $D^e_\beta(n)$ are $\Theta(2^n)$.*

7.2 Sorting by Length-Weighted Reversals

We will now show that the algorithm Bubble Sort is an approximation algorithm for SBBWR and SBWR. We will consider that if $\mathcal{S}$ is a set of rearrangements, then the cost of $\mathcal{S}$ is the sum of the cost of every rearrangement in it and it is denoted by $f(\mathcal{S})$.

We define *2-reversals* as reversals of length 2. Lemma 61 (resp. Lemma 62) shows that, given an optimal solution that costs c for SBBWR (resp. SBWR), it is possible to build a solution that costs at most $d \times c$ and uses only 2-reversals, for some constant d. Note that this does not necessarily give an algorithm to sort binary strings (resp. permutations), because we do not know how to build an optimal solution in the first place. However, this indicates that there is an algorithm that uses only 2-reversals and can guarantee an approximation factor of d. On the other hand, Lemma 63 shows that Bubble Sort is optimal among all algorithms that only use 2-reversals. All these results lead to Corollary 33.

Lemma 61. *There exists a solution $\mathcal{S}'$ that costs at most $1.125f(\mathcal{S})$ and uses only 2-reversals, where $\mathcal{S}$ is an optimal solution that sorts a binary string T with reversals and has cost $f(\mathcal{S})$.*

Proof. Consider a reversal $\rho(i, j)$ from solution $\mathcal{S}$. Assume that it has length ℓ and that it contains k 1's and $\ell - k$ 0's. Note that we can move an element from a position a to a position b with 2-reversals by successively exchanging such element with the element to its right (when $a < b$), which takes an amount of $b - a$ 2-reversals. Also note that we can mimic the result of $\rho(i, j)$ with at most $k(\ell - k)$ 2-reversals, because each of the $\ell - k$ elements 0 will be exchanged with at most k elements 1 (see Example 28).

Now suppose $\mathcal{S}$ can be divided into $\mathcal{S}_1$, which contains all 2-reversals from $\mathcal{S}$, and $\mathcal{S}_2$, which contains the other reversals that have length $\ell \geq 3$. We can build a solution $\mathcal{S}'$ from $\mathcal{S}$ by using the reversals from $\mathcal{S}_1$ and by replacing the reversals from $\mathcal{S}_2$ with 2-reversals as mentioned above. Let $\mathcal{S}'_2$ contain such 2-reversals. Note that $\mathcal{S}'$ contains only 2-reversals and also sorts T. Let ℓ_ρ denote the length of a reversal ρ and k_ρ denote the amount of 1's that ρ contains. We have that

$$\begin{aligned}
f(\mathcal{S}') &= f(\mathcal{S}_1) + f(\mathcal{S}'_2) &&= f(\mathcal{S}_1) + \sum_{\rho \in \mathcal{S}_2} f(2) k_\rho (\ell_\rho - k_\rho) \\
&\leq f(\mathcal{S}_1) + \sum_{\rho \in \mathcal{S}_2} 2^2 \frac{\ell_\rho}{2} \frac{\ell_\rho}{2} &&= f(\mathcal{S}_1) + \sum_{\rho \in \mathcal{S}_2} \ell_\rho^2 \\
&\leq f(\mathcal{S}_1) + \sum_{\rho \in \mathcal{S}_2} 1.125 \times 2^{\ell_\rho} &&= f(\mathcal{S}_1) + 1.125 f(\mathcal{S}_2) \\
&\leq 1.125(f(\mathcal{S}_1) + f(\mathcal{S}_2)) &&= 1.125 f(\mathcal{S})
\end{aligned}$$

where the inequality from the third line is true because $x^2 \leq 1.125(2^x)$ for any $x \geq 3$. □

Example 28. *Let* $T = 011000110111010$. *Reversal* $\rho(5, 14)$ *transforms* T *into* $T' = 0110\overline{10111011000}0$. *We can mimic* $\rho(5, 14)$ *with 2-reversals over* T *in the following manner: consider all elements* 1 *from left to right and move them to the left towards their final position.*

$$\begin{aligned}
T &= 0110\overline{0\underline{1}10111010}\ \text{// move leftmost 1 to position 5} \\
T \leftarrow T{\cdot}\rho(6,7) &= 0110\underline{01}010111010 \\
T \leftarrow T{\cdot}\rho(5,6) &= 0110\overline{10\underline{01}011101}0\ \text{// move leftmost 1 to position 7} \\
T \leftarrow T{\cdot}\rho(7,8) &= 0110\overline{1010\underline{01}11010}\ \text{// move leftmost 1 to position 8} \\
T \leftarrow T{\cdot}\rho(9,10) &= 0110\overline{101\underline{01}011010} \\
T \leftarrow T{\cdot}\rho(8,9) &= 0110\overline{10110\underline{01}11010}\ \text{// move leftmost 1 to position 9} \\
T \leftarrow T{\cdot}\rho(10,11) &= 0110\overline{1011\underline{01}01010} \\
T \leftarrow T{\cdot}\rho(9,10) &= 0110\overline{10111\underline{00}1010}\ \text{// move leftmost 1 to position 11} \\
T \leftarrow T{\cdot}\rho(11,12) &= 0110\overline{101110100\underline{10}}0\ \text{// move leftmost 1 to position 12} \\
T \leftarrow T{\cdot}\rho(13,14) &= 0110\overline{1011101\underline{01}00} \\
T \leftarrow T{\cdot}\rho(12,13) &= 0110\overline{10111011000}0
\end{aligned}$$

Note that each of the six 1*'s present in the segment from 5 to 14 in* T *exchanged position with some of the four* 0*'s and none of them exchanged position with another* 1. *Indeed, at most* 6×4 *2-reversals were made, as mentioned in Lemma 61.*

Lemma 62. *There exists a solution* $\mathcal{S}'$ *that costs at most* $1.5 f(\mathcal{S})$ *and uses only 2-reversals, where* $\mathcal{S}$ *is an optimal solution that sorts a permutation* π *with reversals and has cost* $f(\mathcal{S})$,

Proof. Consider a reversal $\rho(i, j)$ from solution $\mathcal{S}$ and assume that it has length ℓ. Note that we can move an element from a position a to a position $b > a$ with 2-reversals by successively exchanging such element with the element to its right, which takes $b - a$ 2-reversals. In order to mimic the result of reversal $\rho(i, j)$ with 2-reversals, we will: move π_i to position j (this takes $\ell - 1$ 2-reversals and brings π_{i+1} to position i); then move π_{i+1} to position $j - 1$ (this takes $\ell - 2$ 2-reversals and brings π_{i+2} to position i), and so on, until we only need one 2-reversal to move π_{j-1} to position $i + 1$, which would put π_j in position i as well (see Example 29). Therefore, the reversal $\rho(i, j)$ of length ℓ can be mimicked by $\sum_{k=1}^{\ell-1} k = \ell(\ell - 1)/2$ 2-reversals.

Now suppose $\mathcal{S}$ can be divided into $\mathcal{S}_1$, which contains all 2-reversals of $\mathcal{S}$, and $\mathcal{S}_2$, which contains the other reversals that have length $\ell \geq 3$. We can build a solution $\mathcal{S}'$ from

$\mathcal{S}$ by using the reversals from $\mathcal{S}_1$ and by replacing the reversals from $\mathcal{S}_2$ with 2-reversals as mentioned above. Let $\mathcal{S}'_2$ contain such 2-reversals. Note that $\mathcal{S}'$ contains only 2-reversals and sorts π. Let ℓ_ρ denote the length of a reversal ρ. We have that

$$\begin{array}{rclrcl} f(\mathcal{S}') & = & f(\mathcal{S}_1) + f(\mathcal{S}'_2) & = & f(\mathcal{S}_1) + \sum\limits_{\rho\in\mathcal{S}_2} f(2)\dfrac{\ell_\rho(\ell_\rho-1)}{2} \\ & = & f(\mathcal{S}_1) + \sum\limits_{\rho\in\mathcal{S}_2} 2^2\dfrac{\ell_\rho(\ell_\rho-1)}{2} & = & f(\mathcal{S}_1) + \sum\limits_{\rho\in\mathcal{S}_2} 2(\ell_\rho^2-\ell_\rho) \\ & \leq & f(\mathcal{S}_1) + \sum\limits_{\rho\in\mathcal{S}_2} 1.5\times 2^{\ell_\rho} & = & f(\mathcal{S}_1) + 1.5f(\mathcal{S}_2) \\ & \leq & 1.5(f(\mathcal{S}_1)+f(\mathcal{S}_2)) & = & 1.5f(\mathcal{S}) \end{array}$$

where the inequality from the third line is true because $2(x^2-x) \leq 1.5\times 2^x$ for $x>0$. □

Example 29. *Let* $\pi = (9\ 6\ 2\ 14\ 10\ 15\ 7\ 12\ 4\ 11\ 3\ 8\ 13\ 1\ 5)$. *Reversal* $\rho(5,11)$ *transforms* π *into* $\pi' = (9\ 6\ 2\ 14\ \overline{3\ 11\ 4\ 12\ 7\ 15\ 10}\ 8\ 13\ 1\ 5)$. *We can mimic* $\rho(5,11)$ *with 2-reversals over* π *in the following manner: let* $j = 11$*; while element* 3 *is not at position* 5*, move the current element that is in position* 5 *to position* j *and decrement* j*.*

π = (9 6 2 14 $\overline{\underline{10\ 15}\ 7\ 12\ 4\ 11\ 3}$ 8 13 1 5) // *move element* 10 *to position* 11
$\pi \leftarrow \pi\cdot\rho(5,6)$ = (9 6 2 14 $\overline{15\ \underline{10\ 7}\ 12\ 4\ 11\ 3}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(6,7)$ = (9 6 2 14 $\overline{15\ 7\ \underline{10\ 12}\ 4\ 11\ 3}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(7,8)$ = (9 6 2 14 $\overline{15\ 7\ 12\ \underline{10\ 4}\ 11\ 3}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(8,9)$ = (9 6 2 14 $\overline{15\ 7\ 12\ 4\ \underline{10\ 11}\ 3}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(9,10)$ = (9 6 2 14 $\overline{15\ 7\ 12\ 4\ 11\ \underline{10\ 3}}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(10,11)$ = (9 6 2 14 $\overline{\underline{15\ 7}\ 12\ 4\ 11\ 3\ 10}$ 8 13 1 5) // *move element* 15 *to position* 10
$\pi \leftarrow \pi\cdot\rho(5,6)$ = (9 6 2 14 $\overline{7\ \underline{15\ 12}\ 4\ 11\ 3\ 10}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(6,7)$ = (9 6 2 14 $\overline{7\ 12\ \underline{15\ 4}\ 11\ 3\ 10}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(7,8)$ = (9 6 2 14 $\overline{7\ 12\ 4\ \underline{15\ 11}\ 3\ 10}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(8,9)$ = (9 6 2 14 $\overline{7\ 12\ 4\ 11\ \underline{15\ 3}\ 10}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(9,10)$ = (9 6 2 14 $\overline{\underline{7\ 12}\ 4\ 11\ 3\ 15\ 10}$ 8 13 1 5) // *move element* 7 *to position* 9
$\pi \leftarrow \pi\cdot\rho(5,6)$ = (9 6 2 14 $\overline{12\ \underline{7\ 4}\ 11\ 3\ 15\ 10}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(6,7)$ = (9 6 2 14 $\overline{12\ 4\ \underline{7\ 11}\ 3\ 15\ 10}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(7,8)$ = (9 6 2 14 $\overline{12\ 4\ 11\ \underline{7\ 3}\ 15\ 10}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(8,9)$ = (9 6 2 14 $\overline{\underline{12\ 4}\ 11\ 3\ 7\ 15\ 10}$ 8 13 1 5) // *move element* 12 *to position* 8
$\pi \leftarrow \pi\cdot\rho(5,6)$ = (9 6 2 14 $\overline{4\ \underline{12\ 11}\ 3\ 7\ 15\ 10}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(6,7)$ = (9 6 2 14 $\overline{4\ 11\ \underline{12\ 3}\ 7\ 15\ 10}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(7,8)$ = (9 6 2 14 $\overline{\underline{4\ 11}\ 3\ 12\ 7\ 15\ 10}$ 8 13 1 5) // *move element* 4 *to position* 7
$\pi \leftarrow \pi\cdot\rho(5,6)$ = (9 6 2 14 $\overline{11\ \underline{4\ 3}\ 12\ 7\ 15\ 10}$ 8 13 1 5)
$\pi \leftarrow \pi\cdot\rho(6,7)$ = (9 6 2 14 $\overline{\underline{11\ 3}\ 4\ 12\ 7\ 15\ 10}$ 8 13 1 5) // *move element* 11 *to position* 6
$\pi \leftarrow \pi\cdot\rho(5,6)$ = (9 6 2 14 $\overline{3\ 11\ 4\ 12\ 7\ 15\ 10}$ 8 13 1 5)

The following definition will be useful in Lemma 63.

Definition 26. *For a binary string* $T = t_1t_2\ldots t_n$*, the number of* inversions *is defined as*

$$\mathbb{I}(T) = \sum_{1\leq i<j\leq n} I(i,j),\ \textit{where}\ I(i,j) = \begin{cases} 0 & \textit{if } t_i \leq t_j \\ 1 & \textit{otherwise} \end{cases}$$

For a permutation $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$*, the number of* inversions *is defined similarly:*

$$\mathbb{I}(\pi) = \sum_{1\leq i<j\leq n} I(i,j),\ \textit{where}\ I(i,j) = \begin{cases} 0 & \textit{if } \pi_i \leq \pi_j \\ 1 & \textit{otherwise} \end{cases}$$

Lemma 63. *Bubble Sort is optimal among all algorithms that only use 2-reversals.*

Proof. Note that, for either binary strings or permutations, a 2-reversal can decrease the number of inversions by at most one unit. This means that at least $\mathbb{I}(T)$ reversals are needed to sort T and at least $\mathbb{I}(\pi)$ reversals are needed to sort π. Since we are considering only 2-reversals, this means that to sort T or π the cost is at least $f(2)\mathbb{I}(T) = 4\mathbb{I}(T)$ or $4\mathbb{I}(\pi)$, respectively. On the other hand, Bubble Sort only exchanges two adjacent elements if they are in the wrong order, which means that each 2-reversal of Bubble Sort (which costs $f(2)$) decreases the number of inversions by exactly one unit. Therefore, Bubble Sort costs exactly $4\mathbb{I}(T)$ or $4\mathbb{I}(\pi)$ to sort T or π, respectively, that is, it is optimal for sorting using only 2-reversals. □

Corollary 33. *For $f(\ell) = 2^\ell$, Bubble Sort is a* 1.125*-approximation algorithm for* SBBWR *and a* 1.5*-approximation algorithm for* SBWR.

Regarding the diameters $C_r^e(n)$ and $D_r^e(n)$, Theorem 34 shows that they are $\Theta(n^2)$.

Theorem 34. *For $f(\ell) = 2^\ell$, $C_r^e(n)$ and $D_r^e(n)$ are $\Theta(n^2)$.*

Proof. First note that Bubble Sort gives an upper bound of $O(n^2)$: it performs at most $n(n-1)/2$ reversals to sort a permutation π, each of them with a cost of 2^2. Since $C_r^e(n) \leq D_r^e(n)$, this upper bound is valid for both diameters.

The lower bound, given next, uses the function $\mathbb{I}(T)$ from Definition 26, which gives the number of inversions of a bit sequence T.

Let $T' = 1010\ldots10$ and $T'_s = 0^{n/2}1^{n/2}$. It is easy to see that $\mathbb{I}(T'_s) = 0$ and $\mathbb{I}(T') = n(n+2)/8$ (because each 1 of T' forms an inversion with every 0 that appears after it).

Now note that a reversal of length ℓ decreases the number of inversions by at most $\ell(\ell-1)/2$: considering positions i and j that are used by the definition of $\mathbb{I}(T)$, there are exactly $\ell(\ell-1)/2$ pairs (i,j) inside such reversal and one reversal can change the order of elements only if they are affected by it.

Consider an optimal sequence that sorts T' and contains q reversals $\rho_1, \rho_2, \ldots, \rho_q$ of lengths $\ell_1, \ell_2, \ldots, \ell_q$. We know that $\mathbb{I}(T') - \mathbb{I}(T' \cdot \rho_1) \leq \ell_1(\ell_1-1)/2 \leq \ell_1^2$, $\mathbb{I}(T' \cdot \rho_1) - \mathbb{I}(T' \cdot \rho_1 \cdot \rho_2) \leq \ell_2^2$, and so on, until $\mathbb{I}(T' \cdot \rho_1 \cdots \rho_{q-1}) - \mathbb{I}(T'_s) \leq \ell_q^2$. All these together show that

$$\mathbb{I}(T') - \mathbb{I}(T'_s) \leq \sum_{i=1}^{q} \ell_i^2 \leq \sum_{i=1}^{q} 2 \times 2^{\ell_i} = 2\sum_{i=1}^{q} f(\ell_i) = 2d_r^e(T'),$$

because $x^2 \leq 2 \times 2^x$ for all $x \in \mathbb{N}$.

Therefore, $d_r^e(T') \geq (\mathbb{I}(T') - \mathbb{I}(T'_s))/2$, which means that $c_r^e(T')$ is $\Omega(n^2)$ and, thus, $C_r^e(n)$ as well as $D_r^e(n)$ are $\Omega(n^2)$. □

7.3 Sorting by Length-Weighted Transpositions and Sorting by Length-Weighted Reversals and Transpositions

We define *2-transpositions* as transpositions of length 2. Bubble Sort also helps us to find similar results as we gave in Section 7.2 when transpositions are allowed in the rearrangement model, because the result of a 2-reversal is exactly the same as the result of a 2-transposition. We can, therefore, adapt Lemmas 61 and 62 in order to show results for SBBWT, SBBWRT, SBWT, and SBWRT. Lemmas 64 and 65 do that.

Lemma 64. *There exists a solution $\mathcal{S}'$ that costs at most $1.125f(\mathcal{S})$ and uses only 2-transpositions (resp. 2-reversals and 2-transpositions), where $\mathcal{S}$ is an optimal solution that sorts a binary string T with transpositions (resp. reversals and transpositions) and has cost $f(\mathcal{S})$.*

Proof. Lemma 61 already showed how to mimic a reversal with 2-reversals over binary strings. The same analysis can be done for transpositions, because a transposition of length ℓ that contains k 1's and $\ell - k$ 0's can also be mimicked by at most $k(\ell - k)$ 2-transpositions (each of the $\ell - k$ 0's will be exchanged with at most k elements 1's).

The rest of the proof is similar to the proof of Lemma 61. □

Lemma 65. *There exists a solution $\mathcal{S}'$ that costs at most $1.125f(\mathcal{S})$ and uses only 2-transpositions (res. 2-reversals and 2-transpositions), where $\mathcal{S}$ is an optimal solution that sorts a permutation π with transpositions (resp. reversals and transpositions) and has cost $f(\mathcal{S})$.*

Proof. Lemma 62 already showed how to mimic a reversal with 2-reversals over permutations. Now consider a transposition $\tau(i, j, k)$ from solution $\mathcal{S}$ and assume that it has length ℓ, the first segment has length ℓ_1, and the second segment has length ℓ_2 (i.e., $\ell = \ell_1 + \ell_2$). Note that we can move an element from a position a to a position $b > a$ with 2-transpositions by successively exchanging such element with the element to its right, which takes $b - a$ 2-transpositions. In order to mimic the result of transposition $\tau(i, j, k)$ with 2-transpositions, we will: move π_{j-1} to position $k - 1$ (this takes ℓ_2 2-transpositions and does not change the order inside the second segment); then move π_{j-2} to position $k - 2$ (this takes ℓ_2 2-transpositions and also does not change the order inside the second segment); and so on, until we use ℓ_2 2-transpositions to move π_i to position $k - (j - i)$, which would make the whole second segment start at i as well. Therefore, the transposition $\tau(i, j, k)$ of length $\ell = \ell_1 + \ell_2$ can be mimicked by $\ell_1 \times \ell_2$ 2-transpositions. Note that $\ell_1 \times \ell_2 \leq \ell/2 \times \ell/2 = \ell^2/4$.

First we consider SBWT. Suppose $\mathcal{S}$ can be divided into $\mathcal{S}_1$, which contains all 2-transpositions of $\mathcal{S}$, and $\mathcal{S}_2$, which contains the other transpositions that have length $\ell \geq 3$. We can build a solution $\mathcal{S}'$ from $\mathcal{S}$ by using the transpositions from $\mathcal{S}_1$ and by replacing the transpositions from $\mathcal{S}_2$ with 2-transpositions as mentioned above. Let $\mathcal{S}'_2$ contain such 2-transpositions. Note that $\mathcal{S}'$ contains only 2-transpositions and sorts π.

Let ℓ_τ denote the length of a transposition τ. We have that

$$\begin{aligned} f(\mathcal{S}') &= f(\mathcal{S}_1) + f(\mathcal{S}'_2) & &\leq f(\mathcal{S}_1) + \sum_{\tau \in \mathcal{S}_2} f(2)\frac{\ell_\tau^2}{4} \\ &= f(\mathcal{S}_1) + \sum_{\tau \in \mathcal{S}_2} 2^2\frac{\ell_\tau^2}{4} & &= f(\mathcal{S}_1) + \sum_{\tau \in \mathcal{S}_2} \ell_\tau^2 \\ &\leq f(\mathcal{S}_1) + \sum_{\tau \in \mathcal{S}_2} 1.125 \times 2^{\ell_\tau} & &= f(\mathcal{S}_1) + 1.125 f(\mathcal{S}_2) \\ &\leq 1.125(f(\mathcal{S}_1) + f(\mathcal{S}_2)) & &= 1.125 f(\mathcal{S}) \end{aligned}$$

where the inequality from the third line is true because $x^2 \leq 1.125(2^x)$ for any $x \geq 3$.

We can show the same result for SBWRT with very similar arguments. □

These two lemmas along with Lemma 63 (which can be directly adapted to 2-transpositions) lead to the following corollary. Theorem 36 shows that the diameters of the problems considered in this section are also $\Theta(n^2)$.

Corollary 35. *For $f(\ell) = 2^\ell$, Bubble Sort is a 1.125-approximation algorithm for the problems* SBBWT, SBBWRT, SBWT, *and* SBWRT.

Theorem 36. *For $f(\ell) = 2^\ell$, $C_t^e(n)$, $C_{rt}^e(n)$, $D_t^e(n)$, and $D_{rt}^e(n)$ are $\Theta(n^2)$.*

Proof. Similar to Theorem 34. □

7.4 Sorting Signed Permutations and Signed Binary Strings by Length-Weighted Prefix Rearrangements

We also start by giving a lower bound on the distances in Lemma 66.

Lemma 66. *Let $\beta \in \{p\bar{r}, p\bar{r}t\}$. For any signed permutation π and signed binary string T with n valid elements,*

$$c_\beta^e(T) \geq 2^n \text{ and } d_\beta^e(\pi) \geq 2^n.$$

Proof. Similar to the proof of Lemma 60. □

We use here the same idea presented in Section 7.1, with some adaptations. The algorithms to sort binary strings first place the rightmost 1 or -1 together with the last block of 1's until we have only one block of 1's in the end; after this, we may still have -0 at some positions in the beginning of the string, so the algorithms fix the sign of the rightmost -0 until we have only positive 0's. The algorithms to sort permutations always place the highest element that is out of order (in absolute value) in its correct position.

Algorithms 33 and 34 show the algorithms for SBBWP$\bar{\text{R}}$ and SBWP$\bar{\text{R}}$, respectively, while Algorithms 35 and 36 show the algorithms for SBBWP$\bar{\text{R}}$T and SBWP$\bar{\text{R}}$T, respectively. Theorem 37 shows that such algorithms are $(4+n/2^{n-1})$-approximation algorithms for the four problems being considered in this chapter.

Theorem 37. *For $f(\ell) = 2^\ell$,* SBBWP$\bar{\text{R}}$, SBBWP$\bar{\text{R}}$T, SBWP$\bar{\text{R}}$, *and* SBWP$\bar{\text{R}}$T *are $(4 + n/2^{n-1})$-approximable.*

Proof. Note that for both permutations and binary strings, any of the Algorithms 33, 35, 34, or 36 need at most three rearrangements at each iteration of their while loops. Consider that, at each iteration, the algorithms will place an element at position y. As a high estimative, we can consider that every possible position will be considered, so $1 \leq y \leq n$. Note that position y may itself contain the element that we want to place there (it can be negative). Also note that once an element is placed at position y, such position will never be considered again by the algorithms. Therefore, these algorithms will sort a binary string or a permutation with n valid elements with a cost of at most

$$f(1) + \sum_{i=2}^{n}(f(i) + f(1) + f(i)) = 2 + \sum_{i=2}^{n}(2^{i+1} + 2) = 4 \times 2^n + 2n - 8 < 4 \times 2^n + 2n,$$

which, according to the lower bound given in Lemma 66, gives the approximation factor of $(4 \times 2^n + 2n)/2^n = 4 + n/2^{n-1}$. □

Algorithm 33 A $(4 + n/2^{n-1})$-approximation algorithm for SBBWP$\bar{\text{R}}$.

4-BWP$\bar{\text{R}}$ (T, n)

Input: binary string T and its size n
Output: cost used to sort T

$c \leftarrow 0$
while T does not have only one block of 1's in the end **do**
  Let y be the smallest position such that $t_i = 1$ for all $y \leq i \leq n + 1$
  Let $x < y$ be the highest position such that $t_x = \pm 1$
  if $x \neq 1$ **then** // Bring t_x to the beginning if necessary
    $T \leftarrow T \cdot \bar{\rho}_p(x)$
    $c \leftarrow c + f(x)$
  if $t_1 \neq -1$ **then** // The first element should be a -1
    $T \leftarrow T \cdot \bar{\rho}_p(1)$
    $c \leftarrow c + f(1)$
  // Put the first element next to the last block (of 1's)
  $T \leftarrow T \cdot \bar{\rho}_p(y)$
  $c \leftarrow c + f(y)$
while there is at least one -0 in T **do**
  Let y be the highest position such that $t_y = -0$
  if $y \neq 1$ **then**
    $T \leftarrow T \cdot \bar{\rho}_p(y - 1)$
    $c \leftarrow c + f(y - 1)$
  if there is another -0 in T **then**
    $T \leftarrow T \cdot \bar{\rho}_p(1) \cdot \bar{\rho}_p(y)$
    $c \leftarrow c + f(1) + f(y)$
return c

Similarly to what happened for the unsigned problems, these algorithms given so far show that $C^e_\beta(n)$ and $D^e_\beta(n)$ are both $O(2^n)$, for $\beta \in \{p\bar{r}, p\bar{r}t\}$. The lower bounds given in Lemma 66, on the other hand, show that $C^e_\beta(n)$ and $D^e_\beta(n)$ are both $\Omega(2^n)$, which leads us directly to the following theorem.

Theorem 38. *For $\beta \in \{p\bar{r}, p\bar{r}t\}$ and $f(\ell) = 2^\ell$, $C^e_\beta(n)$ and $D^e_\beta(n)$ are $\Theta(2^n)$.*

Example 30. *The following example shows the execution of Algorithm 33 over* $T = -0\,-1\,-1\,+0\,-0\,-0\,+1\,+1\,+0\,+1\,-1\,-1\,+0\,+1\,-0$. *The first part leaves* T *with a unique block of* $+1$*'s in the end:*

$$
\begin{aligned}
T &= \underline{-0-1-1+0-0-0+1+1+0+1-1-1+0+1}-0 && \text{// bring rightmost } +1 \text{ to beginning}\\
T \leftarrow T\cdot\bar{\rho}_p(14) &= \underline{-1-0+1+1-1-0-1-1+0+0-0+1+1+0-0} && \text{// } t_1 \text{ is negative, so put it in the end}\\
T \leftarrow T\cdot\bar{\rho}_p(15) &= \underline{+0-0-1-1+0-0-0+1+1+0+1-1-1}+0+1 && \text{// bring rightmost } -1 \text{ to beginning}\\
T \leftarrow T\cdot\bar{\rho}_p(13) &= \underline{+1+1-1-0-1-1+0+0-0+1+1+0-0+0}+1 && \text{// } t_1 \text{ is positive, so reverse it}\\
T \leftarrow T\cdot\bar{\rho}_p(1) &= \underline{-1+1-1-0-1-1+0+0-0+1+1+0-0+0}+1 && \text{// and put it next to the last block}\\
T \leftarrow T\cdot\bar{\rho}_p(14) &= \underline{-0+0-0-1-1+0-0-0+1+1+0+1-1+1}+1 && \text{// bring rightmost } -1 \text{ to beginning}\\
T \leftarrow T\cdot\bar{\rho}_p(13) &= \underline{+1-1-0-1-1+0+0-0+1+1+0-0+0}+1+1 && \text{// } t_1 \text{ is positive, so reverse it}\\
T \leftarrow T\cdot\bar{\rho}_p(1) &= \underline{-1-1-0-1-1+0+0-0+1+1+0-0+0}+1+1 && \text{// and put it next to the last block}\\
T \leftarrow T\cdot\bar{\rho}_p(13) &= \underline{-0+0-0-1-1+0-0-0+1+1+0+1+1}+1+1 && \text{// bring rightmost } +1 \text{ (not in last block) to beginning}\\
T \leftarrow T\cdot\bar{\rho}_p(10) &= \underline{-1-1+0+0-0+1+1+0-0+0+0}+1+1+1+1 && \text{// } t_1 \text{ is negative, so put it next to the last block}\\
T \leftarrow T\cdot\bar{\rho}_p(11) &= \underline{-0-0+0-0-1-1+0-0-0+1+1}+1+1+1+1+1 && \text{// bring rightmost } -1 \text{ to beginning}\\
T \leftarrow T\cdot\bar{\rho}_p(6) &= \underline{+1+1+0-0+0+0+0-0-0+1+1+1+1+1+1} && \text{// } t_1 \text{ is positive, so reverse it}\\
T \leftarrow T\cdot\bar{\rho}_p(1) &= \underline{-1+1+0-0+0+0+0-0-0}+1+1+1+1+1+1+1 && \text{// and put it next to the last block}\\
T \leftarrow T\cdot\bar{\rho}_p(9) &= \underline{+0+0-0-0-0+0-0-1}+1+1+1+1+1+1+1+1 && \text{// bring rightmost } -1 \text{ to beginning}\\
T \leftarrow T\cdot\bar{\rho}_p(8) &= \underline{+1+0-0+0+0+0-0-0}+1+1+1+1+1+1+1+1 && \text{// } t_1 \text{ is positive, so reverse it}\\
T \leftarrow T\cdot\bar{\rho}_p(1) &= \underline{-1+0-0+0+0+0-0-0}+1+1+1+1+1+1+1+1 && \text{// and put it next to the last block}\\
T \leftarrow T\cdot\bar{\rho}_p(8) &= +0+0-0-0-0+0-0+1+1+1+1+1+1+1+1 && \text{//}
\end{aligned}
$$

The second part leaves T *with a unique block of* $+0$*: in the beginning:*

$$
\begin{aligned}
T &= \underline{+0+0-0-0-0+0-0}+1+1+1+1+1+1+1+1 && \text{// bring the rightmost } -0 \text{ to beginning}\\
T \leftarrow T\cdot\bar{\rho}_p(7) &= \underline{+0-0+0+0+0-0-0}+1+1+1+1+1+1+1+1 && \text{// there are other } -0\text{'s, so reverse it}\\
T \leftarrow T\cdot\bar{\rho}_p(1) &= \underline{-0-0+0+0+0-0-0}+1+1+1+1+1+1+1+1 && \text{// and put it back in its original position}\\
T \leftarrow T\cdot\bar{\rho}_p(7) &= \underline{+0+0-0-0-0+0+0}+1+1+1+1+1+1+1+1 && \text{// bring the rightmost } -0 \text{ to beginning}\\
T \leftarrow T\cdot\bar{\rho}_p(5) &= \underline{+0+0+0-0-0}+0+0+1+1+1+1+1+1+1+1 && \text{// there are other } -0\text{'s, so reverse it}\\
T \leftarrow T\cdot\bar{\rho}_p(1) &= \underline{-0+0+0-0-0}+0+0+1+1+1+1+1+1+1+1 && \text{// and put it back in its original position}\\
T \leftarrow T\cdot\bar{\rho}_p(5) &= \underline{+0+0-0-0+0}+0+0+1+1+1+1+1+1+1+1 && \text{// bring the rightmost } -0 \text{ to beginning}\\
T \leftarrow T\cdot\bar{\rho}_p(4) &= \underline{+0+0-0-0}+0+0+0+1+1+1+1+1+1+1+1 && \text{// there are other } -0\text{'s, so reverse it}\\
T \leftarrow T\cdot\bar{\rho}_p(1) &= \underline{-0+0-0-0}+0+0+0+1+1+1+1+1+1+1+1 && \text{// and put it back in its original position}\\
T \leftarrow T\cdot\bar{\rho}_p(4) &= \underline{+0+0-0+0}+0+0+0+1+1+1+1+1+1+1+1 && \text{// bring the rightmost } -0 \text{ to beginning}\\
T \leftarrow T\cdot\bar{\rho}_p(3) &= \underline{+0-0-0}+0+0+0+0+1+1+1+1+1+1+1+1 && \text{// there are other } -0\text{'s, so reverse it}\\
T \leftarrow T\cdot\bar{\rho}_p(1) &= \underline{-0-0-0}+0+0+0+0+1+1+1+1+1+1+1+1 && \text{// and put it back in its original position}\\
T \leftarrow T\cdot\bar{\rho}_p(3) &= \underline{+0+0+0}+0+0+0+0+1+1+1+1+1+1+1+1
\end{aligned}
$$

Algorithm 34 A $(4 + n/2^{n-1})$-approximation algorithm for SBWP$\bar{\text{R}}$.

4-WP$\bar{\text{R}}$(π, n)

Input: permutation π and its size n
Output: number of operations used to sort π

```
1  c ← 0
2  while π ≠ ι_n do
3      Let π_j = i be the highest element such that π_|i| ≠ |i|
4      if j ≠ 1 then        // Bring the element to the beginning if necessary
5          π ← π · ρ̄_p(j)
6          c ← c + f(j)
7      if π_1 > 0 then      // The element should be negative
8          π ← π · ρ̄_p(1)
9          c ← c + f(1)
       // Move i to its right position
10     π ← π · ρ̄_p(|i|)
11     c ← c + f(|i|)
12 return c
```

Algorithm 35 A $(4 + n/2^{n-1})$-approximation algorithm for SBBWP$\bar{\text{R}}$T.

4-BWP$\bar{\text{R}}$T(T, n)

Input: binary string T and its size n
Output: cost used to sort T

$c \leftarrow 0$
while T does not have only one block of 1's in the end **do**
  Let y be the smallest position such that $t_i = 1$ for all $y \leq i \leq n + 1$
  Let $x < y$ be the highest position such that $t_x = \pm 1$
  if $t_x = 1$ **then**
    $T \leftarrow T \cdot \tau_p(x + 1, y)$
    $c \leftarrow c + f(y - 1)$
  else
    if $x \neq 1$ **then**
      // Bring t_x to the beginning if necessary
      $T \leftarrow T \cdot \bar{\rho}_p(x)$
      $c \leftarrow c + f(x)$
    $T \leftarrow T \cdot \bar{\rho}_p(1) \cdot \bar{\rho}_p(y - 1)$
    $c \leftarrow c + f(1) + f(y - 1)$
if T is sorted **then**
  return c
while T does not have only one block of -0's in the beginning **do**
  Let y be the highest position such that $t_y = -0$
  Let x be the first position of the block that contains t_y
  // Bring such block to the beginning
  $T \leftarrow T \cdot \tau_p(x, y + 1)$
  $c \leftarrow c + f(y)$
Let x be the last position of the first block (of -0's)
$T \leftarrow T \cdot \bar{\rho}_p(x)$
$c \leftarrow c + f(x)$
return c

Algorithm 36 A $(4 + n/2^{n-1})$-approximation algorithm for SBWP$\bar{\text{R}}$T.

4-WP$\bar{\text{R}}$T(π, n)

Input: permutation π and its size n
Output: cost used to sort π

$c \leftarrow 0$
while $\pi \neq \iota_n$ **do**
  Let $\pi_j = i$ be the highest element such that $\pi_{|i|} \neq |i|$
  if $\pi_j > 0$ **then**
    $\pi \leftarrow \pi \cdot \tau_p(j + 1, i + 1)$
    $c \leftarrow c + f(i)$
  else
    if $j \neq 1$ **then** // Bring the element to the beginning if necessary
      $\pi \leftarrow \pi \cdot \bar{\rho}_p(j)$
      $c \leftarrow c + f(j)$
    $\pi \leftarrow \pi \cdot \bar{\rho}_p(1) \cdot \bar{\rho}_p(|i|)$
    $c \leftarrow c + f(1) + f(|i|)$
return c

7.5 Summary of the Chapter

Table 7.1 summarizes the best approximation factors and bounds for the diameters of the problems that were studied in this chapter.

Table 7.1: Summary of the results obtained for length-weighted rearrangement problems when $f(\ell) = 2^\ell$.

Rearrangements	**Approximation Factor**		**Diameter**	
	Bin. Str.	**Perm.**	**Bin. Str.**	**Perm.**
Pref. Reversals	3 (Thm. 28)	3 (Thm. 30)	$\Theta(2^n)$ (Thm. 32)	$\Theta(2^n)$ (Thm. 32)
Pref. Transpositions and Pref. Reversals and Transp.	2 (Thm. 29)	2 (Thm. 31)	$\Theta(2^n)$ (Thm. 32)	$\Theta(2^n)$ (Thm. 32)
Reversals	1.125 (Cor. 33)	1.5 (Cor. 33)	$\Theta(n^2)$ (Thm. 34)	$\Theta(n^2)$ (Thm. 34)
Transpositions and Reversals and Transpositions	1.125 (Cor. 35)	1.125 (Cor. 35)	$\Theta(n^2)$ (Thm. 36)	$\Theta(n^2)$ (Thm. 36)
Sig. Pref. Reversals and Sig. Pref. Reversals and Transp.	$4+\frac{n}{2^{n-1}}$ (Thm. 37)	$4+\frac{n}{2^{n-1}}$ (Thm. 37)	$\Theta(2^n)$ (Thm. 38)	$\Theta(2^n)$ (Thm. 38)

Chapter 8

Final Considerations

This thesis presented the most important results obtained during the period of the doctorate. For all problems considered, there has been an initial and extensive study of them. For the traditional approach, we considered a total of 10 problems and developed algorithms and bounds for the distance and diameter for 6 of them. For the length-weighted approach, we considered a total of 13 problems and also developed algorithms and bounds for the distance and diameter for all of them. We also considered a new cost function over 7 of the studied problems. All results obtained are summarized in Sections 4.9, 6.9, and 7.5.

We were able to produce the following contributions to the academic area during this period:

1. a paper entitled "Sorting Permutations by Prefix and Suffix Versions of Reversals and Transpositions" [44], presented during the Latin American Theoretical Informatics Symposium in 2014;

2. a paper entitled "On Sorting of Signed Permutations by Prefix and Suffix Reversals and Transpositions" [45], presented during the International Conference on Algorithms for Computational Biology in 2014;

3. a paper entitled "On the Diameter of Rearrangement Problems" [46], also presented during the International Conference on Algorithms for Computational Biology in 2014;

4. a paper entitled "Approximation Algorithms for Sorting by Length-Weighted Prefix and Suffix Operations" [47], published in the journal Theoretical Computer Science in 2015.

We also contributed with the paper entitled "A General Heuristic for Genome Rearrangement Problems" [22], published in the Journal of Bioinformatics and Computational Biology in 2014. This paper presents an heuristic that receives a sorting sequence and tries to decrease its size by using the optimal sorting sequence of small permutations.

The first important future work is developing good lower bounds for the distance and approximation algorithms for the case where $\alpha > 0$ when considering both prefix and suffix rearrangements. Another possible future direction is trying to improve the

approximation factors for sorting by transpositions and sorting by prefix rearrangements when $\alpha = 1$ in the same way that was done for sorting by reversals, or trying to improve the results for signed permutations and binary strings by designing specific algorithms for such problems. Furthermore, many questions remain unanswered concerning the complexities of the problems we mentioned: for $\alpha = 0$ some of them were already shown to be NP-hard, for $\alpha = 1$ SBBWR is polynomial, and for $\alpha \geq 3$ SBWR and SBBWR are polynomial; all other cases remain open.

Bibliography

[1] D. A. Bader, B. M. E. Moret, and M. Yan. A Linear-Time Algorithm for Computing Inversion Distance Between Signed Permutations with an Experimental Study. *Journal of Computational Biology*, 8:483–491, 2001.

[2] V. Bafna and P. A. Pevzner. Genome Rearrangements and Sorting by Reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996.

[3] V. Bafna and P. A. Pevzner. Sorting by Transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240, 1998.

[4] C. Baudet, U. Dias, and Z. Dias. Length and Symmetry on the Sorting by Weighted Inversions Problem. In S. Campos, editor, *Advances in Bioinformatics and Computational Biology*, volume 8826 of *Lecture Notes in Computer Science*, pages 99–106. Springer International Publishing, Switzerland, 2014.

[5] M. A. Bender, D. Ge, S. He, H. Hu, R. Y. Pinter, S. S. Skiena, and F. Swidan. Improved Bounds on Sorting by Length-Weighted Reversals. *Journal of Computer and System Sciences*, 74(5):744–774, 2008.

[6] A. Bergeron. A Very Elementary Presentation of the Hannenhalli-Pevzner Theory. *Discrete Applied Mathematics*, 146(2):134–145, 2005.

[7] P. Berman, S. Hannenhalli, and M. Karpinski. 1.375-Approximation Algorithm for Sorting by Reversals. In R. Möhring and R. Raman, editors, *Proceedings of the 10th Annual European Symposium on Algorithms (ESA'2002)*, volume 2461 of *Lecture Notes in Computer Science*, pages 200–210. Springer-Verlag Berlin Heidelberg New York, Berlin/Heidelberg, Germany, 2002.

[8] M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric Genome Rearrangement. *Gene*, 172(1):GC11–GC17, 1996.

[9] L. Bulteau, G. Fertin, and I. Rusu. Sorting by Transpositions is Difficult. *SIAM Journal on Computing*, 26(3):1148–1180, 2012.

[10] L. Bulteau, G. Fertin, and I. Rusu. Pancake Flipping is Hard. *Journal of Computer and System Sciences*, 81(8):1556–1574, 2015.

[11] A. Caprara. Sorting Permutations by Reversals and Eulerian Cycle Decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110, 1999.

[12] X. Chen. On Sorting Unsigned Permutations by Double-Cut-and-Joins. *Journal of Combinatorial Optimization*, 25(3):339–351, 2013.

[13] B. Chitturi. Tighter Upper Bound for Sorting Permutations with Prefix Transpositions. *Theoretical Computer Science*, 602:22–31, 2015.

[14] B. Chitturi and I. H. Sudborough. Bounding Prefix Transposition Distance for Strings and Permutations. *Theoretical Computer Science*, 421:15–24, 2012.

[15] B. Chitturi, W. Fahle, Z. Meng, L. Morales, C. O. Shields, I. H. Sudborough, and W. Voit. An (18/11)n Upper Bound for Sorting by Prefix Reversals. *Theoretical Computer Science*, 410(36):3372–3390, 2009.

[16] D. A. Christie. A 3/2-Approximation Algorithm for Sorting by Reversals. In H. Karloff, editor, *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'1998)*, pages 244–252, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[17] J. Cibulka. On Average and Highest Number of Flips in Pancake Sorting. *Theoretical Computer Science*, 412(8-10):822–834, 2011.

[18] D. S. Cohen and M. Blum. On the Problem of Sorting Burnt Pancakes. *Discrete Applied Mathematics*, 61(2):105–120, 1995.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, London, England, 3rd edition, 2009.

[20] T. da S. Arruda, U. Dias, and Z. Dias. Heuristics for the Sorting by Length-Weighted Inversions Problem on Signed Permutations. In A.-H. Dediu, C. Martín-Vide, and B. Truthe, editors, *Algorithms for Computational Biology*, volume 8542 of *Lecture Notes in Computer Science*, pages 59–70. Springer International Publishing, Switzerland, 2014.

[21] U. Dias and Z. Dias. Heuristics for the Transposition Distance Problem. *Journal of Bioinformatics and Computational Biology*, 11(5):17, 2013.

[22] U. Dias, G. R. Galvão, C. N. Lintzmayer, and Z. Dias. A General Heuristic for Genome Rearrangement Problems. *Journal of Bioinformatics and Computational Biology*, 12(3):26, 2014.

[23] Z. Dias and U. Dias. Sorting by Prefix Reversals and Prefix Transpositions. *Discrete Applied Mathematics*, 181:78–89, 2015.

[24] Z. Dias and J. Meidanis. Sorting by Prefix Transpositions. In A. H. F. Laender and A. L. Oliveira, editors, *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE'2002)*, volume 2476 of *Lecture Notes in Computer Science*, pages 65–76. Springer-Verlag Berlin Heidelberg New York, Berlin/Heidelberg, Germany, 2002.

[25] H. Dweighter. Problem E2569. *American Mathematical Monthly*, 82:1010, 1975.

[26] I. Elias and T. Hartman. A 1.375-Approximation Algorithm for Sorting by Transpositions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):369–379, 2006.

[27] H. Eriksson, K. Eriksson, J. Karlander, L. Svensson, and J. Wastlund. Sorting a Bridge Hand. *Discrete Mathematics*, 241(1-3):289–300, 2001.

[28] G. Fertin, A. Labarre, I. Rusu, É. Tannier, and S. Vialette. *Combinatorics of Genome Rearrangements*. Computational Molecular Biology. The MIT Press, London, England, 2009.

[29] G. Fertin, L. Jankowiak, and G. Jean. Prefix and Suffix Reversals on Strings. In C. Iliopoulos, S. Puglisi, and E. Yilmaz, editors, *String Processing and Information Retrieval*, volume 9309 of *Lecture Notes in Computer Science*, pages 165–176. Springer International Publishing, Switzerland, 2015.

[30] J. Fischer and S. W. Ginzinger. A 2-Approximation Algorithm for Sorting by Prefix Reversals. In G. S. Brodal and S. Leonardi, editors, *Proceedings of the 13th Annual European Conference on Algorithms (ESA'2005)*, volume 3669 of *Lecture Notes in Computer Science*, pages 415–425, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[31] G. R. Galvão and Z. Dias. An Audit Tool for Genome Rearrangement Algorithms. *Journal of Experimental Algorithmics*, 19:1–34, 2014.

[32] G. R. Galvão, C. Baudet, and Z. Dias. Sorting Signed Circular Permutations by Super Short Reversals. In R. Harrison, Y. Li, and I. Măndoiu, editors, *Proceedings of the 11th International Symposium on Bioinformatics Research and Applications (ISBRA'2015)*, Lecture Notes in Computer Science, pages 272–283. Springer International Publishing, Switzerland, 2015.

[33] G. R. Galvão, O. Lee, and Z. Dias. Sorting Signed Permutations by Short Operations. *Algorithms for Molecular Biology*, 10(1):1–17, 2015.

[34] W. H. Gates and C. H. Papadimitriou. Bounds for Sorting by Prefix Reversal. *Discrete Mathematics*, 27(1):47–57, 1979.

[35] Q.-P. Gu, S. Peng, and I. H. Sudborough. A 2-Approximation Algorithm for Genome Rearrangements by Reversals and Transpositions. *Theoretical Computer Science*, 210 (2):327–339, 1999.

[36] Y. Han. Improving the Efficiency of Sorting by Reversals. In H. R. Arabnia and H. Valafar, editors, *Proceedings of the 2006 International Conference on Bioinformatics & Computational Biology (BIOCOMP'2006)*, pages 635–646. CSREA Press, 2006.

[37] S. Hannenhalli and P. A. Pevzner. Transforming Cabbage into Turnip: Polynomial Algorithm for Sorting Signed Permutations by Reversals. *Journal of the ACM*, 46 (1):1–27, 1999.

[38] T. Hartman and R. Sharan. A 1.5-Approximation Algorithm for Sorting by Transpositions and Transreversals. *Journal of Computer and System Sciences*, 70(3):300–320, 2005.

[39] M. Hasan, A. Rahman, M. K. Rahman, M. S. Rahman, M. Sharmin, and R. Yeasmin. Pancake Flipping and Sorting Permutations. *Journal of Discrete Algorithms*, 33:139–149, 2015.

[40] M. H. Heydari and I. H. Sudborough. On the Diameter of the Pancake Network. *Journal of Algorithms*, 25(1):67–94, 1997.

[41] J. D. Kececioglu and D. Sankoff. Exact and Approximation Algorithms for Sorting by Reversals, with Application to Genome Rearrangement. *Algorithmica*, 13:180–210, 1995.

[42] A. Labarre. Edit Distances and Factorisations of Even Permutations. In D. Halperin and K. Mehlhorn, editors, *Proceedings of the 16th Annual European Symposium on Algorithms (ESA'2008)*, volume 5193 of *Lecture Notes in Computer Science*, pages 635–646, Berlin, Heidelberg, 2008. Springer-Verlag.

[43] G.-H. Lin and G. Xue. Signed Genome Rearrangement by Reversals and Transpositions: Models and Approximations. *Theoretical Computer Science*, 259(1-2):513–531, 2001.

[44] C. N. Lintzmayer and Z. Dias. Sorting Permutations by Prefix and Suffix Versions of Reversals and Transpositions. In A. Pardo and A. Viola, editors, *LATIN 2014: Theoretical Informatics*, volume 8392 of *Lecture Notes in Computer Science*, pages 671–682. Springer Berlin Heidelberg, Heidelberg, Germany, 2014.

[45] C. N. Lintzmayer and Z. Dias. On Sorting of Signed Permutations by Prefix and Suffix Reversals and Transpositions. In A.-H. Dediu, C. Martín-Vide, and B. Truthe, editors, *Algorithms for Computational Biology*, volume 8542 of *Lecture Notes in Computer Science*, pages 146–157. Springer International Publishing, Switzerland, 2014.

[46] C. N. Lintzmayer and Z. Dias. On the Diameter of Rearrangement Problems. In A.-H. Dediu, C. Martín-Vide, and B. Truthe, editors, *Algorithms for Computational Biology*, volume 8542 of *Lecture Notes in Computer Science*, pages 158–170. Springer International Publishing, Switzerland, 2014.

[47] C. N. Lintzmayer, G. Fertin, and Z. Dias. Approximation Algorithms for Sorting by Length-Weighted Prefix and Suffix Operations. *Theoretical Computer Science*, 593: 26–41, 2015.

[48] J. Meidanis, M. E. M. T. Walter, and Z. Dias. Transposition Distance Between a Permutation and its Reverse. In R. Baeza-Yates, editor, *Proceedings of the 4th South American Workshop on String Processing (WSP'1997)*, pages 70–79, Ontario, Canada, 1997. Carleton University Press.

[49] T. C. Nguyen, H. T. Ngo, and N. B. Nguyen. Sorting by Restricted-Length-Weighted Reversals. *Genomics Proteomics & Bioinformatics*, 3(2):120–127, 2005.

[50] R. Y. Pinter and S. Skiena. Genomic Sorting with Length-Weighted Reversals. *Genome Informatics*, 13:2002, 2002.

[51] A. Rahman, S. Shatabda, and M. Hasan. An Approximation Algorithm for Sorting by Reversals and Transpositions. *Journal of Discrete Algorithms*, 6(3):449–457, 2008.

[52] I. Rusu. Log-Lists and Their Applications to Sorting by Transpositions, Reversals and Block-Interchanges. http://arxiv.org/abs/1507.01512, 2015.

[53] M. Sharmin, R. Yeasmin, M. Hasan, A. Rahman, and M. S. Rahman. Pancake Flipping with Two Spatulas. *Electronic Notes in Discrete Mathematics*, 36:231–238, 2010.

[54] F. Swidan, M. A. Bender, D. Ge, S. He, H. Hu, and R. Y. Pinter. Sorting by Length-Weighted Reversals: Dealing with Signs and Circularity. In S. Sahinalp, S. Muthukrishnan, and U. Dogrusoz, editors, *Combinatorial Pattern Matching*, volume 3109 of *Lecture Notes in Computer Science*, pages 32–46. Springer Berlin Heidelberg, Heidelberg, Germany, 2004.

[55] E. Tannier, A. Bergeron, and M.-F. Sagot. Advances on Sorting by Reversals. *Discrete Applied Mathematics*, 155(6-7):881–888, 2007.

[56] M. E. M. T. Walter, Z. Dias, and J. Meidanis. Reversal and Transposition Distance of Linear Chromosomes. In *Proceedings of the 5th International Symposium on String Processing and Information Retrieval (SPIRE'1998)*, pages 96–102, Los Alamitos, CA, USA, 1998. IEEE Computer Society.