Improved Bounds on Sorting by Length-Weighted Reversals *

Michael A. Bender[†‡] Dongdong Ge[§] Simai He[¶] Haodong Hu[†] Ron Y. Pinter[‖]
Steven Skiena[†] Firas Swidan[‖]

Abstract

We study the problem of sorting binary sequences and permutations by length-weighted reversals. We consider a wide class of cost functions, namely $f(\ell) = \ell^\alpha$ for all $\alpha \geq 0$, where ℓ is the length of the reversed subsequence. We present tight or nearly tight upper and lower bounds on the worst-case cost of sorting by reversals. Then we develop algorithms to approximate the optimal cost to sort a given input. Furthermore, we give polynomial-time algorithms to determine the optimal reversal sequence for a restricted but interesting class of sequences and cost functions. Our results have direct application in computational biology to the field of comparative genomics.

1 Introduction

In the problem of *sorting by reversals (SBR)* we are given as input a permutation to sort. Our only allowed operation is a *reversal* of a segment of contiguous elements by which we inverse their sequential order. The problem of sorting by reversals arises in comparative genomics, where the elements of the permutation are genes and reversal (or inversion) *mutations* occur frequently in the evolution of chromosomes. The minimum-cost reversal distance[1] is a useful measure for reconstructing the evolutionary history of an organism because the most parsimonious series[2] of reversals transforming one sequence to another corresponds to a possible evolutionary path between the two organisms. This analysis has been applied, for example, to drosophila [13, 27], plants [3, 21], viruses [14], and mammals [12, 23].

Traditionally [5, 19], such analysis assumes that each reversal has unit cost independent of the length of the fragment reversed. However, the mechanics of genome rearrangements suggest that the frequencies of reversals can be dependent on fragment length [25]. Preliminary results on genome rearrangements that assign a length-dependent cost to reversals appear in [9], and these results indicate that length indeed plays an important role in biasing certain rearrangement patterns.

*An earlier version of this paper appeared in the Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2004 [6].

[†]Department of Computer Science, SUNY Stony Brook, Stony Brook, NY 11794-4400, USA. Email: {bender, huhd, skiena}@cs.sunysb.edu.

[‡]Supported in part by Sandia National Laboratories and NSF Grants ACI-032497, CCR-0208670, EIA-0112849, CCF-0621439/0621425, CCF-0540897/05414009, CCF-0634793/0632838, and CNS-0627645.

[§]Department of Management Science and Engineering, Stanford University, Stanford, CA 94305, USA. Email: dongdong@stanford.edu

[¶]Department of System Engineering and Engineering Management, Chinese University of Hong Kong, Hong Kong , China. Email: smhe@se.cuhk.edu.hk

[‖]Department of Computer Science, Technion – Israel Institute of Technology, Haifa 32000, Israel. Email: {firas, pinter}@cs.technion.ac.il.

[1]The problems of sorting a given permutation by reversals and finding the reversal distance between two given permutations are equivalent: simply relabel the elements of the target permutation to be the identity and use the same relabeling for the source permutation.

[2]*i.e.*, the series requiring least evolutionary efforts.

In this paper we consider the problem of sorting by reversals, where the cost of a reversal is a function $f(\ell)$ of its length ℓ. Our objective is to minimize the total cost of the reversals performed during the sort. We analyze two classes of problems:

- *Worst-Case Sorting* — We give upper and lower bounds on the cost to sort permutations of length n.
- *Input-Specific Sorting* — We give exact and approximation algorithms to determine the minimum cost to sort a given permutation.

Pinter and Skiena [24] were the first to study worst-case and input-specific sorting by length-weighted costs. Specifically, they consider costs given by linear weight function $f(\ell) = \ell$. They give an $O(n \lg^2 n)$ bound for sorting and an $O(\lg^2 n)$-approximation algorithm for sorting a given permutation.

1.1 Results

In this paper we consider the problem of sorting by reversals, generalizing to a wide class of cost functions, namely $f(\ell) = \ell^\alpha$ for $\alpha \geq 0$. In addition to permutations, we also consider 0/1 sequences as inputs. Permutations are relevant for genome-rearrangement studies, where orthologous[3] genes in a pair of organisms are represented as a permutation. Algorithms for 0/1 sequences, intrinsically interesting on their own, are used as subroutines in the algorithms for input permutations.

The family of cost functions is general enough to include unit costs ($\alpha = 0$), *additive* costs, where $f(x) + f(y) = f(x + y)$ ($\alpha = 1$), *subadditive* costs, where $f(x) + f(y) > f(x + y)$ ($\alpha < 1$), and *superadditive* costs, where $f(x) + f(y) < f(x + y)$ ($\alpha > 1$).

We present the following results, which are summarized in Table 1:

- We prove an $\Omega(n \lg n)$ lower bound on sorting for additive cost functions. This is the first non-trivial lower bound on sorting by length-weighted reversals, and it holds and is tight even for 0/1 sequences.
- More generally, we prove tight or near-tight bounds on sorting for all $\alpha \geq 0$, as summarized in Table 1. Specifically, we give lower bounds for all $\alpha \geq 0$ and show that one algorithm matches or nearly matches these bounds.
- We give approximation algorithms for all $\alpha \geq 0$, also summarized in Table 1. In contrast to the sorting bounds, different algorithms are needed to achieve approximation guarantees for each of the additive, subadditive, and superadditive cases.
- For linear cost functions, we give a polynomial-time algorithm for optimally sorting a 0/1 sequence. We use this result to give an $O(\lg n)$ approximation algorithm for sorting permutations, improving the $O(\lg^2 n)$ result from [24].

1.2 Previous Work

The problem of computing the reversal distance between two permutations and its applications to comparative genomics have received extensive attention over the last decade. There are two variants of the problem: the *unsigned* case, in which we disregard the orientation of the elements throughout the reversal process, and the *signed* case, where the directions of the elements do matter. Both measures have merit in terms of the underlying biology. Moreover, the existence of circular genomes (*e.g.*, prokaryotic) as well as non-circular genomes (*e.g.*, mammalian) motivates developing algorithms for handling both cases. In this paper we focus on the non-circular unsigned case. Extensions of these results to other configurations were

[3] Orthologous genes evolved from the same ancestral gene, and so induce a one-to-one mapping.

α Value	Lower Bounds	Upper Bounds		Approximation Ratio	
		Permutations	0/1's	Permutations	0/1's
$0 \leq \alpha < 1$	$\Omega(n)$	$O(n \lg n)$	$\Theta(n)$		$O(1)$
$\alpha = 1$	$\Omega(n \lg n)$	$O(n \lg^2 n)$	$\Theta(n \lg n)$	$O(\lg n)$	1
$1 < \alpha < 2$	$\Omega(n^\alpha)$	$\Theta(n^\alpha)$	$\Theta(n^\alpha)$	$O(\lg n)$	$O(1)$
$\alpha \geq 2$	$\Omega(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	2	1

Table 1: Sorting Bounds (Lower and Upper) and Approximation Ratios for 0/1 sequences and integer permutations.

considered in [28]. Note, however, that the low-cost of single-element reversals means that our solutions apply to the signed case when $\alpha \geq 1$ — see [28].

For the case of unit-cost ($\alpha = 0$), unsigned reversals, the problem of computing the reversal distance has been shown to be NP-complete by Caprara [10]; our problem, in which the cost depends on the length of the subsequence being reversed, inherits hardness for $\alpha = 0$ from this result. Kececloglu and Sankoff [19] give approximation algorithms on reversal distance that guarantee a ratio at most 2 times optimal, which Bafna and Pevzner [5] improved to a factor of 7/4 approximation; recently, Berman, Hannenhalli, and Karpinski [8] reduced this factor even further, to 1.375. Kececloglu and Sankoff [20] report on the success of heuristics and search in determining the reversal distance for chromosomes.

In a celebrated result Hannenhalli and Pevzner [16] gave a polynomial-time algorithm for the case of unit-cost, signed reversals. An elementary exposition of the Hannenhalli-Pevzner theory appears in [7]. Recently, Siepel [26] gave an efficient algorithm for constructing/enumerating *all* minimum-length reversal sequences. The huge number of such sequences implies that other criteria must be employed to have hope of reconstructing the true evolutionary history. Ajana et al. [1] developed algorithms for users of a (signed) reversal algorithm to choose one or several possible solutions based on different criteria, including additive reversal costs; this flexibility was shown to be useful for testing certain reversal hypotheses.

Minimum-cost unsigned reversal sorting has also been studied from the other end of the cost spectrum, under models where the cost increases so dramatically with length that only length-2 reversals can be afforded. Hence, each reversal simply transposes adjacent elements. Bubble-sort and insertion sort [22] both sort any permutation π using exactly one transposition for each inversion in π, thus minimizing the number of reversals.

Outline. The rest of the paper is organized as follows. In Section 2 we define the notation used throughout the paper. Section 3 presents upper and lower bounds for the problem of sorting any permutation and any 0/1 sequence. In Section 4 we give polynomial-time algorithms to optimally sort a given 0/1 sequence when $\alpha = 1$ and in Section 5 we prove their correctness. Section 6 provides algorithms for sorting a specific permutation or a specific 0/1 sequence for $0 \leq \alpha \leq 2$. In Section 6.4 we handle the case of $2 < \alpha$. We conclude in Section 7 with a summary and open problems.

2 Notation

We refer to the permutation version of sorting by reversals as *PSBR* (*Permutation Sorting By Reversals*), and we refer to sorting 0/1 sequences by reversals as *BSBR* (*Binary Sorting By Reversals*). In the following we present notation and terminology used in the discussion of these two versions.

2.1 Notation for 0/1 Sequences

Consider a sequence $T = t_1, \ldots, t_n$, for $t_i \in \{0, 1\}$. Refer to T as a *0/1 bit sequence*. Denote the length of T by $|T|$ (i.e $|T| = n$). The sequence T is called *sorted* if it consists of a single consecutive 0's subsequence followed by a single consecutive 1's subsequence.

A reversal $\rho = \rho(i, j)$, for $i < j$, transforms a bit sequence

$$T = t_1, t_2, \ldots, t_{i-1}, t_i, t_{i+1}, \ldots, t_{j-1}, t_j, t_{j+1}, \ldots, t_n$$

to a bit sequence

$$T \cdot \rho(i, j) = t_1, t_2, \ldots, t_{i-1}, t_j, t_{j-1}, \ldots, t_{i+1}, t_i, t_{j+1}, \ldots, t_n.$$

A reversal series $\rho_1, \ldots, \rho_m$ is called a *sorting reversal series* of a 0/1 bit sequence T, if $T \cdot \rho_1 \cdots \rho_m$ is a sorted 0/1 bit sequence.

Given an arbitrary cost function $f : \mathbb{R}^+ \to \mathbb{R}^+$, the cost of a reversal $\rho(i, j)$ equals the function applied to the reversal's length. We overload the notation and denote the cost of a reversal ρ by $f(\rho) = f(j - i + 1)$.

The cost of a reversal series $\varrho = \rho_1, \ldots, \rho_m$ equals the sum of the costs of all reversals, that is $f(\varrho) = f(\rho_1, \ldots, \rho_m) = \sum_{i=1}^{m} f(\rho_i)$. Our goal is to find a sorting reversal series having the minimum cost, *i.e.*, a *minimum sorting series*. We denote the minimum cost by $\mathrm{opt}(\cdot)$.

Given a bit sequence T, a reversal ρ affecting it, and a subsequence T_1 of T, we denote the restriction of ρ to T_1 by $\rho|T_1$. The cost of the restriction is denoted by $f(\rho|T_1)$.

Given a bit sequence $T = t_1, \ldots, t_n$, refer to a maximal contiguous subsequence of only 1's or only 0's as a *block*. The *weight* of a block is the number of bits in it. To standardize the representation of sequences, we assume that the leading block in a sequence is a 0-block and that the closing block in a sequence is a 1-block. Both the leading and the closing blocks might have a 0-weight. Thus, a bit sequence T with $g + 1$ blocks of 0's and $g + 1$ blocks of 1's can be represented as a sequence of blocks $b = b(T) = 0^{w_0}1^{w_1}0^{w_2} \cdots 0^{w_{2g}}1^{w_{2g+1}}$ (the *block sequence*) or as a sequence of weights $w = w(T) = w_0, \ldots, w_{2g+1}$ (the *weighted sequence*); see Figure 1.

$T =$	000	11	0000000	111111	0000	11111
$b =$	0^3	1^2	0^7	1^6	0^4	1^5
$w =$	3	2	7	6	4	5

Figure 1: An example of a bit sequence and the corresponding block and weighted representations.

Each weighted sequence $w = w_0, \ldots, w_{2g+1}$ is naturally associated with a bit sequence, which we denote by $T = T(w)$. The weighted sequence w is called *sorted* if its associated bit sequence $T(w)$ is sorted, which only happens if $g = 0$.

Let $t_p, \ldots, t_q$ be a subsequence of T and let $u_i, w_{i+1}, \ldots, w_{j-1}, u_j$, for $0 < u_i \leq w_i$ and $0 < u_j \leq w_j$, be the subsequence of weights corresponding to it. The effect of a reversal ρ affecting the subsequence of weights on w is defined by mimicking it on the bit sequence, *i.e.*, $w \cdot \rho = w(T(w) \cdot \rho')$, where $\rho' = \rho|t_p, \ldots, t_q$. If $u_i = w_i$ and $u_j = w_j$, we call $w_i, \ldots, w_j$ a *segment* of w. A reversal affecting the segment $w_i, \ldots, w_j$ is denoted by $\rho(i, j)$; see Figure 2.

For $i, j \in \mathbb{N}$, write $i \equiv j$ if $i \equiv j \pmod 2$. Note that if $i \equiv 0$, then w_i corresponds to a 0-block and vice versa.

2.2 Notation for Permutations

Given a permutation π, denote its median by s. If all elements smaller (greater) than the median are located to its left (right), we call the permutation *separated*. Given a permutation π that is not separated, we refer to a reversal series ϱ that separates π, *i.e.*, $\pi \cdot \varrho$ is separated, as a *separating* series.

b	$=$	0^3	1^2	$[0^7$	$1^6]$	0^4	1^5			w	$=$	3	2	[7	6]	4	5
$b \cdot \rho(2,3)$	$=$	0^3	1^8	0^{11}	1^5					$w \cdot \rho(2,3)$	$=$	3	8	11	5		

(a) (b)

Figure 2: A reversal affecting the 0/1 sequence from Figure 1. The reversal is indicated by brackets [,]. (a) The reversal's effect on the block representation of the sequence. (b) The reversal's effect on the weighted representation of the sequence.

3 Sorting Bounds

Pinter and Skiena [24] proved that $O(n \lg n)$ is an upper bound for BSBR when the cost function is linear. Based on this result they showed that $O(n \lg^2 n)$ is an upper bound for PSBR when the cost function is linear.

In this section, we generalize their results to a wide range of cost functions, namely, $f(\ell) = \ell^\alpha$, for $\alpha \geq 0$. In addition, we show that the upper bounds are tight or nearly tight for the whole range. Formally, let $\mathcal{C}_T(n) = \max\{\text{opt}(T) : |T| = n\}$ and $\mathcal{C}_\pi(n) = \max\{\text{opt}(\pi) : |\pi| = n\}$ denote the worst-case cost for sorting a 0/1 sequence T and a permutation π of length n. We give lower and upper bounds for the functions $\mathcal{C}_T(n)$ and $\mathcal{C}_\pi(n)$.

3.1 Upper Bounds for $0 \leq \alpha < 2$

We first give a divide-and-conquer algorithm for the BSBR problem. We then use this algorithm as a subroutine in an algorithm for the PSBR problem. We analyze both algorithms for all $0 \leq \alpha < 2$.

To sort a 0/1 sequence, recursively sort its left and right halves. This step results in a sequence with block representation $0^k 1^i 0^j 1^l$. To complete the sort, perform one more reversal of the subsequence $1^i 0^j$. See Algorithm 1 (`ZerOneSort_DivideConquer`) for the pseudocode.

To sort a permutation π, first separate it. Then recursively separate the left (elements smaller than the median) and the right (rest of elements) halves of the permutation. To perform the separation, let s be the median of π, and map $\pi = \pi_1 \cdots \pi_n$ to a 0/1 sequence $T = t_1, \ldots, t_n$, such that for all $i = 1 \ldots n$,

$$t_i = \begin{cases} 0, & \text{if } \pi_i < s, \\ 1, & \text{otherwise .} \end{cases}$$

See Algorithm 2 (permTo01) for the pseudocode. Let ϱ be the sorting series resulting from applying Algorithm 1 to T. To separate π, perform ϱ on it. See Algorithm 3 (`PermutationSort_DivideConquer`) for the pseudocode.

Algorithm 1 `ZerOneSort_DivideConquer` (T)

```
1: if T is sorted then
2:     return 0
3: else
4:     c1 ← ZerOneSort_DivideConquer (t_1, ..., t_⌊n/2⌋)
5:     c2 ← ZerOneSort_DivideConquer (t_⌊n/2⌋+1, ..., t_n)
6:     i ← #_1(t_1, ..., t_⌊n/2⌋)
7:     j ← #_0(t_⌊n/2⌋+1, ..., t_n)
8:     return c1 + c2 + f(i + j)
9: end if
```

Theorem 1. *The following upper bounds hold for $\mathcal{C}_T(n)$ and $\mathcal{C}_\pi(n)$ under the cost functions $f(\ell) = \ell^\alpha$, for $0 \le \alpha < 2$.*

$$\mathcal{C}_T(n) = \begin{cases} O(n), & 0 \le \alpha < 1, \\ O(n \lg n), & \alpha = 1, \\ O(n^\alpha), & 1 < \alpha < 2, \end{cases} \tag{1}$$

and

$$\mathcal{C}_\pi(n) = \begin{cases} O(n \lg n), & 0 \le \alpha < 1, \\ O(n \lg^2 n), & \alpha = 1, \\ O(n^\alpha), & 1 < \alpha < 2. \end{cases} \tag{2}$$

Proof. The algorithms `ZerOneSort_DivideConquer` and `PermutationSort_DivideConquer` imply the following recurrences for $C_T(n)$ and for $C_\pi(n)$:

$$\begin{aligned} \mathcal{C}_T(n) &\le 2\mathcal{C}_T(n/2) + O(n^\alpha) \\ \mathcal{C}_\pi(n) &\le 2\mathcal{C}_\pi(n/2) + \mathcal{C}_T(n) \ . \end{aligned}$$

Solving these recurrences, we achieve the stated bounds. □

Algorithm 2 permTo01(π)

```
1: s ← median of π
2: for i = 1 to |π| do
3:    if π_i < s then
4:       T_i ← 0
5:    else
6:       T_i ← 1
7:    end if
8: end for
9: return T
```

Algorithm 3 `PermutationSort_DivideConquer` (π)

```
1: T ← permTo01(π)
2: c ← ZerOneSort_DivideConquer (T), with separating series ϱ
3: π ← π · ϱ
4: p_1 ← PermutationSort_DivideConquer (π_1, ..., π_⌊n/2⌋)
5: p_2 ← PermutationSort_DivideConquer (π_⌊n/2⌋+1, ..., π_n)
6: return c + p_1 + p_2
```

3.2 Lower Bounds for $0 \le \alpha < 2$

We show that the upper bounds for BSBR are tight. To do so, we introduce potential functions and prove that they are lower bounds on the sorting cost (we use different potential functions for each α-subrange). Then we find hard BSBR instances to establish the lower bounds. Since $\mathcal{C}_T(n) \le \mathcal{C}_\pi(n)$, the lower bounds of BSBR hold as well for PSBR.

Lower Bound for $\alpha = 1$

We show that the cost to sort n elements by reversals with a linear cost function ($\alpha = 1$) is $\Omega(n \lg n)$, even when all elements are zeros and ones.

Theorem 2. *For linear cost function $f(\ell) = \ell$ both $\mathcal{C}_T(n)$ and $\mathcal{C}_\pi(n)$ are in $\Omega(n \lg n)$.*

We use the potential-function argument to prove the lower bound on the cost to sort the sequence $T = 0101 \cdots 01$ by reversals.

Before the sorting begins, we match the ith 0 with the ith 1. Throughout the algorithm we keep this matching, and we let d_i be the current distance between the ith 0 and ith 1 after some reversals. The potential function is

$$P(T) = \sum_{i=1}^{n/2} \lg d_i.$$

Lemma 3. *The initial value of the potential function for the sequence $T = 0101 \cdots 01$ is 0, and the final value is $\Omega(n \lg n)$.*

Proof. Initially $d_i = 1$ for all i, which implies that the potential function is 0.

To give the lower bound on the final value, consider the $n/4$ 0's at the left end of the sorted sequence $0^{n/2}1^{n/2}$. The distance d_i between each of these 0's and its partner 1's is at least $n/4$. So the value of the potential function $P(T)$ for the final sequence $0^{n/2}1^{n/2}$ is at least $(n/4)\lg(n/4)$, establishing the bound. □

We show how a reversal affects the value of d_i in the potential function by considering the ith 0/1-pair.

Observation 4. *The distance d_i only changes when one element of the ith 0/1-pair is inside the reversal and the other is outside it.*

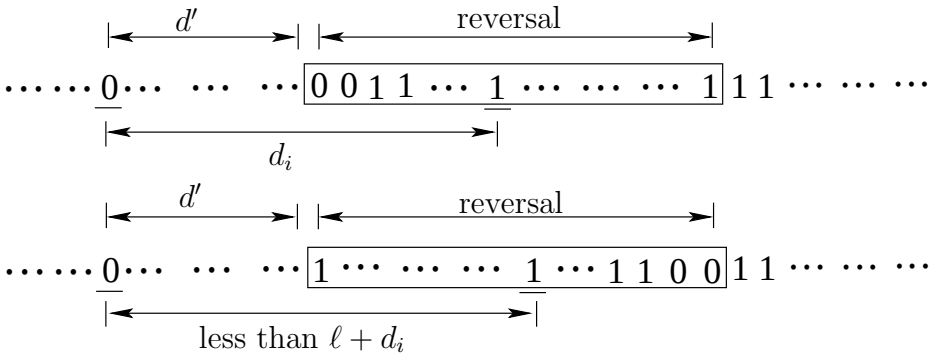

Figure 3: The sequence before and after one reversal.

Lemma 5. *A reversal of length ℓ increases the potential function $P(T)$ by at most 4ℓ.*

Proof. Suppose that for a reversal of length ℓ, one element of the ith 0/1-pair is inside the reversal and the other one is outside it, so that d_i is affected by the reversal. The new distance between those two elements can increase to at most $d_i + \ell$ because each element in the reversal is moved at most the distance ℓ.

Without loss of generality, assume that the 0 is outside the reversed sequence and the 1 is inside it. Suppose that the distance from the 0 to the beginning of the reversed sequence is d'; see Figure 3. Then the contribution of the distance of this pair d_i to the potential function is less than $\lg(\ell + d_i) - \lg d_i =$

$\lg(1+\ell/d_i) \le \lg(1+\ell/d')$. The distance d' must be a natural number, and the same value of d' occurs at most twice in one reversal, once on the left and once on the right side of the reversed sequence.

By Observation 4, there are at most ℓ such pairs increasing the value of the potential function. Therefore, the value increases by at most

$$\begin{aligned} 2\sum_{j=1}^{\ell/2} \lg(1+\ell/j) &\le 2\sum_{j=1}^{\ell/2} (1+\lg(\ell/j)) \\ &\le \ell + 2\sum_{j=1}^{\ell} \lg(\ell/j) \\ &= \ell + 2\lg(\ell^\ell/\ell!). \end{aligned}$$

By Stirling's formula, $\ell^\ell/\ell! \le e^\ell$ for $\ell \ge 1$. Therefore $\lg(\ell^\ell/\ell!) \le \ell \lg e \le \frac{3}{2}\ell$. So the value of the potential function increases by at most $\ell + 3\ell = 4\ell$. □

By combining Lemmas 3 and 5, we establish Theorem 2.

Lower Bound for $1 < \alpha < 2$

We now give a lower bound of $\Omega(n^\alpha)$ on BSBR and PSBR for $1 < \alpha < 2$.

Theorem 6. *For cost functions $f(\ell) = \ell^\alpha$, where $1 < \alpha < 2$, both $\mathcal{C}_T(n)$ and $\mathcal{C}_\pi(n)$ are in $\Omega(n^\alpha)$.*

The proof follows a potential-function argument and is similar to the proof of Theorem 2. Specifically, we show that sorting the sequence $0101\cdots01$ of length n requires cost $\Omega(n^\alpha)$.

Before the sorting begins, we match the ith 0 with the ith 1. Throughout the algorithm we keep this matching and we let d_i be the current distance between the ith 0 and ith 1 after some reversals. We define the potential function to be

$$P(T) = \sum_{i=1}^{n/2} d_i^{\alpha-1}.$$

Lemma 7. *The initial value of the potential function is $\Theta(n)$. The final value of the potential function is $\Omega(n^\alpha)$.*

Proof. Initially $d_i = 1$, which implies that the potential function is $\Theta(n)$. To bound the final value, consider the first half of the 0's at the left side of the final sequence. The distance between each of these 0's and its partner 1's is at least $n/4$. So the value of the potential function $P(T)$ for the final sequence $0^{n/2}1^{n/2}$ is at least $(n/4)(n/4)^{\alpha-1}$, thus establishing the bound. □

We now show how a reversal affects the value of the potential function.

Lemma 8. *A reversal of length ℓ increases the potential function by at most $2\ell^\alpha$.*

Proof. Suppose that for a reversal of length ℓ, one element of the ith 0/1-pair is inside the reversal and the other one is outside, so that d_i is affected by the reversal. The new distance between those two elements can increase to at most $d_i + \ell$ because each element in the reversal is moved at most the distance ℓ.

Without loss of generality, assume that the 0 is outside the reversed sequence and the 1 is inside it. Suppose that the distance from the 0 to the beginning of the reversed sequence is d'; see Figure 3. Then the contribution of the distance of this pair d_i to the potential function is at most $(\ell + d')^{\alpha-1} - (d')^{\alpha-1}$.

Note that the function $x^{\alpha-2}$ is a decreasing function if $\alpha < 2$. By the Intermediate-Value Theorem, we obtain

$$(\ell + d')^{\alpha-1} - (d')^{\alpha-1} = (\alpha-1)\ell(d' + \xi\ell)^{\alpha-2} \le (\alpha-1)\ell\,(d')^{\alpha-2},$$

where $\xi \in [0,1]$.

The distance d' must be a natural number, and the same value of d' can occur at most twice, once for the left side and once for the right side of the reversed sequence. By Observation 4, there are at most ℓ such pairs increasing the value of the potential function.

Therefore the value of the potential function increases by at most $2\sum_{j=1}^{\ell/2}(\alpha-1)\ell\, j^{\alpha-2}$. We bound the sum by an integral and evaluate the integral:

$$\begin{aligned} 2\sum_{j=1}^{\ell/2}(\alpha-1)\ell\, j^{\alpha-2} &\leq 2\ell \int_{x=0}^{\ell/2+1} dx\,(\alpha-1)x^{\alpha-2} \\ &= 2\ell(\ell/2+1)^{\alpha-1} \leq 2\ell\cdot\ell^{\alpha-1} = 2\,\ell^{\alpha}, \end{aligned}$$

that is, the value of potential function increases by at most $2\ell^{\alpha}$. □

We obtain the following corollary directly by noting that the cost to reverse a sequence of length ℓ is ℓ^{α}.

Corollary 9. *If a given reversal increases the potential function by Δ, then the cost of the reversal is at least $\Delta/2$.*

Proof. Assume that the reversal's length is ℓ. Then its cost is $f(\ell) = \ell^{\alpha}$, and from Lemma 8, this reversal increases the potential function by at most $2\ell^{\alpha}$. □

Theorem 6 follows directly from Corollary 9.

Lower Bound for $0 \leq \alpha < 1$

The lower bound is $\Omega(n)$ on both BSBR and PSBR when $0 \leq \alpha < 1$. Our results are tight for 0/1 sequences, but there is a logarithmic gap for permutations:

Theorem 10. *For cost functions $f(\ell) = \ell^{\alpha}$, where $0 \leq \alpha < 1$, both $\mathcal{C}_T(n)$ and $\mathcal{C}_\pi(n)$ are in $\Omega(n)$.*

Proof. As before, we provide a lower bound on the cost to sort the sequence $T = t_1, t_2, \cdots, t_n = 0101\cdots01$ of length n.

We use a potential-function argument. We define the potential function

$$P(T) = \sum_{i=1}^{n-1} |\, t_i - t_{i+1}\,|.$$

The initial value of the potential function P is $n-1$, and after sorting, the its final value is 1. Moreover, each reversal can change its value by at most 2. Thus, we need at least $n/2-1$ reversals, each of length greater or equal to 2, to sort T. Therefore the sorting cost is at least $\Omega(2^{\alpha}n) = \Omega(n)$. □

3.3 Lower and Upper Bounds for $\alpha \geq 2$

Sorting by reversals is straightforward when $\alpha \geq 2$ because the problem can be solved asymptotically optimally using bubble sort; it is never worth reversing sequences of length greater than 2. Thus, we have the following theorem for $\alpha \geq 2$:

Theorem 11. *For cost functions $f(\ell) = \ell^{\alpha}$, where $\alpha \geq 2$, $C_T(n) = C_\pi(n) = \Theta(n^2)$.*

Proof. Bubble-Sort immediately gives an $O(n^2)$ upper bound on the sorting cost. The $\Omega(n^2)$ lower bound follows a potential-function argument. Consider a 0/1 sequence $T = t_1, t_2, \ldots, t_n$. For any two elements t_i and t_j, we say that they are *in correct order* if $t_i \leq t_j$ and $i < j$. We define an order function $X(i, j)$ over pairs of elements (t_i, t_j) to be

$$X(i,j) = \begin{cases} 0, & t_i \leq t_j \text{ and } i < j, \\ 1, & \text{otherwise.} \end{cases}$$

We define the potential function to be the number of out-of-order pairs:

$$P(T) = \sum_{1 \leq i < j \leq n} X(i,j).$$

Consider the sequence $1010\cdots10$ of length n. For this sequence, the initial value of the potential is $n(2+n)/8$, and the sorted sequence is $000\cdots0111\cdots1$ with the potential value 0.

The cost of a reversal of length ℓ is at least ℓ^2. However, such a reversal decreases the potential value by at most $\ell(\ell-1)/2$, because there are exactly $\ell(\ell-1)/2$ pairs in the reverse sequence, and the reversal can only change the orientation of pairs of elements included in the reversed sequence. Therefore, $\Omega(n^2)$ is a lower bound. □

4 Polynomial-Time Algorithms for 0/1 Sorting for $\alpha = 1$

In this section we give an exact algorithm for solving the BSBR problem when $\alpha = 1$. The idea is based on restricting the set of candidate solutions by characterizing the properties of optimal ones. Then a search is performed on the restricted set by means of dynamic-programming in polynomial-time. The four characterizing properties are introduced in this section. The proofs of the corresponding lemmas are given in Section 5. After introducing the properties, we give a naïve $O(n^4)$ algorithm then improve its running time to $O(n^3)$.

Throughout this section, unless mentioned otherwise, the cost function f is linear.

4.1 Basic Properties

We now introduce the properties required for proving the correctness of the algorithms. For sake of simplicity, we introduce the properties in terms of bit sequences. Equivalent definitions by means of weighted sequences can be easily derived and are used in the sequel.

A reversal $\rho = \rho(i, j)$ acting on a 0/1 bit sequence $T = t_1, \ldots, t_n$ is said to *affect* an element t_k of T if $i \leq k \leq j$. The reversal ρ is said to affect a subsequence or a block of T, if it affects all their elements.

Definition 12 (Trivial Reversal). Let $\rho = \rho(i, j)$ be a reversal acting on a 0/1 bit sequence $T = t_1, \ldots, t_n$. The reversal ρ is called *trivial* if it affects the block of leading 0's or the block of closing 1's in T.

For example, the reversal $0\overbrace{0111}^{\rho}001100011$ affects the leading 0's block and is thus trivial.

Definition 13 (Useless Reversal). Let $\rho = \rho(i, j)$ be a reversal acting on a 0/1 bit sequence $T = t_1, \ldots, t_n$. The reversal ρ is called *useless* if $t_i = t_j$.

For example, the reveral $001110\overbrace{0110}^{\rho}0011$ affects a subsequence starting and ending with a 0 and is thus useless.

Definition 14 (Cutting Reversal). Let $\rho = \rho(i, j)$ be a reversal acting on a 0/1 bit sequence $T = t_1, \ldots, t_n$. The reversal ρ is called *cutting* if either $i \geq 2$ and $t_{i-1} = t_i$ or $j \leq n-1$ and $t_j = t_{j+1}$.

For example, the reversal $001110\overbrace{011}^{\rho}00011$ cuts through a block of 0's and is thus cutting.

Definition 15 (Complex Reversal). Let $\rho = \rho(i,j)$ be a reversal acting on a 0/1 bit sequence. The reversal ρ is called *complex* if ρ affects more than 2 blocks. Otherwise ρ is called *simple*.

For example, the reversal $00111\overbrace{001100011}^{\rho}$ affects four blocks and is hence complex. On the other hand the reversal $00111\overbrace{0011}^{\rho'}00011$ affects exactly 2 blocks and is thus simple. In addition, notice that ρ' is neither trivial, useless, nor cutting. We refer to such a reversal as a *good* reversal.

Definition 16 (Good Reversal). Let $\rho = \rho(i,j)$ be a reversal acting on a 0/1 bit sequence. The reversal ρ is called *good* if ρ is neither trivial, useless, cutting, nor complex.

In Section 5 we show that a reversal in an optimal reversal series is neither trivial, useless, nor cutting (Lemmas 25, 27 and 28). In addition, we show that there exists an optimal series containing no complex reversals (Lemma 37). Thus, we get the following theorem.

Theorem 17. *There exists an optimal reversal series in which all reversals are good.*

The proofs of the above lemmas and the discussion in this section are based on a characterization of the sorting cost by means of *reversal counts*, *i.e.*, the number of reversals in which each element of the bit series takes part.

Definition 18 (Reversal Count). Given a reversal series $\rho_1, \ldots, \rho_m$ acting on a 0/1 bit sequence $T = t_1, \ldots, t_n$, denote the number of reversals in which element t_i participates by $N(t_i)$ (notice that the element t_i might change its location while applying the reversal series). We call $N(t_i)$ the *reversal count* of t_i. If the series $\rho_1, \ldots, \rho_m$ does not cut a subsequence $t_i, \ldots, t_j$ of T, we define the reversal count of the subsequence as the number of reversals in which the subsequence takes part and denote it by $N(t_i, \ldots, t_j)$.

The following equation (applying for linear cost functions — $\alpha = 1$) relating the reversal counts to the reversal series cost says that we can measure the cost per reversal or per bit in the sequence:

$$\sum_{i=1}^{m} f(\rho_i) = \sum_{j=1}^{n} f(N(t_j)) \ . \tag{3}$$

Given a weighted sequence $w_0, \ldots, w_{2g+1}$ and a reversal series $\rho_1, \ldots, \rho_m$ define $w^k = w \cdot \rho_1 \cdots \rho_k$ for $1 \le k \le m$ to be the result of applying the first k reversals on w; thus $w^0 = w$.

4.2 A Polynomial-Time Algorithm

Given a 0/1 weighted sequence $w = w_0, \ldots, w_{2g+1}$, denote the set of all sorting series that contains only good reversals by $\mathscr{S}$. By Theorem 17, the set $\mathscr{S}$ contains a minimum sorting series. We show here that it is possible to find such a series in polynomial-time.

The following lemmas characterize the sorting process under good reversals. They are used for calculating the optimal cost in polynomial-time as well as for proving that a minimum reversal series containing no complex reversals exists (see Section 5.2). In the discussion to come, given a weighted sequence $w = w_0, \ldots, w_{2g+1}$ and a reversal series ϱ containing no cutting reversals, we define c_i to be the number of reversals in which w_i takes part, *i.e.*, $c_i = N(w_i)$.

Lemma 19. *Let $w = w_0, \ldots, w_{2g+1}$ be a weighted sequence and let $\rho_1, \ldots, \rho_m$ be a good sorting series. Then each weight of w^k for $0 \le k \le m$ contains a weight of w that takes part during $\rho_1, \ldots, \rho_k$ in zero reversals.*

Proof. By induction on k. Base case: the claim is trivial for $k = 0$. Induction step: suppose each weight of w^k contains a weight of w that takes part during $\rho_1, \ldots, \rho_k$ in zero reversals. We need to prove that each weight of w^{k+1} contains a weight of w that takes part during $\rho_1, \ldots, \rho_{k+1}$ in zero reversals. Since ρ_{k+1} is good, it must be of the form $\rho(i-1, i)$ for some i, $2 \leq i \leq 2g_k$, where $w^k = w_0, \ldots, w_{2g_k+1}$. Thus, it unifies w_i^k with w_{i-2}^k, and w_{i-1}^k with w_{i+1}^k. By the induction assumption, w_{i-2}^k contains a weight of w, denote it v, that is not affected by $\rho_1, \ldots, \rho_k$. Since ρ_{k+1} does not affect the weight w_{i-2}^k, the same weight v of w is not affected by $\rho_1, \ldots, \rho_{k+1}$. Thus, the unification of w_i^k with w_{i-2}^k contains a weight of w (*i.e.*, v) that is not affected by $\rho_1, \ldots, \rho_{k+1}$. A similar argument holds for the union of w_{i-1}^k with w_{i+1}^k. The other weights of w^k are not affected by ρ_{k+1}, and therefore they contain by the induction assumption a weight of w that takes part during $\rho_1, \ldots, \rho_k$ in zero reversals. The same weight takes part during $\rho_1, \ldots, \rho_{k+1}$ in zero reversals as well. □

Lemma 20. *Let $w = w_0, \ldots, w_{2g+1}$ be a weighted sequence and let $\varrho = \rho_1, \ldots, \rho_m$ be a good sorting series. There exist indices i and j such that $j \equiv 0$, $i \equiv 1$, $c_i = c_j = 1$, and $i < j$.*

Proof. Consider $w^{m-1} = w \cdot \rho_1 \cdots \rho_{m-1} = w_0^{m-1}, w_1^{m-1}, w_2^{m-1}, w_3^{m-1}$. The sequence w^{m-1} must contain four weights since ρ_m sorts it. Furthermore, ρ_m must affect w_1^{m-1} and w_2^{m-1}. By Lemma 19 each of w_1^{m-1} and w_2^{m-1} contains a weight of w that takes part during $\rho_1, \ldots, \rho_{m-1}$ in zero reversals. Denote the indices of these weights in w by i and j, respectively. Thus we get $i \equiv 1$, $j \equiv 0$, and $c_i = c_j = 1$. In addition, since w_i and w_j take part during $\rho_1, \ldots, \rho_{m-1}$ in zero reversals, the weights in which w_i and w_j are contained in w^{m-1}, that is w_1^{m-1} and w_2^{m-1} respectively, have a relative order identical to that of w_i and w_j in w. Hence, we get $i < j$. □

Lemma 21. *Let w, ϱ, i, and j be as in Lemma 20, and let $w^{m-1} = w \cdot \rho_1 \cdots \rho_{m-1} = w_0^{m-1}, w_1^{m-1}, w_2^{m-1}, w_3^{m-1}$. Then $\rho_m = \rho(1, 2)$ and $w_1^{m-1} = \sum_{k \in \{1,3,\ldots,j-1\}} w_k$ and $w_2^{m-1} = \sum_{r \in \{i+1, i+3, \ldots, 2g\}} w_r$.*

Proof. For each $k \in \{1, 3, \ldots, j-1\}$, the weight w_k can change its relative order to the weight w_j only when ρ_m is performed, because w_j takes part only in that reversal. A similar claim holds for w_r where $r \in \{i+1, i+3, \ldots, 2g\}$ and w_i. Hence, the reversal ρ_m must affect all these blocks to change their relative order. □

Lemma 21 provides a way for finding a sorting series having the minimum cost in $\mathscr{S}$: For all pairs (i, j), such that $i < j$, $i \equiv 1$, and $j \equiv 0$, consider the optimal sorting cost in which $c_i = c_j = 1$ and take the minimum over all (i, j) pairs.

Finding the optimal sorting cost in which $c_i = c_j = 1$ can be done with dynamic programming. Define a dynamic programming matrix $A(i, j, b)$ as follows: cell $A(i, j, b)$ contains the optimal value of separating the segment (i, j) of w with polarity b, where $b = 0$ corresponds to a positive polarity (*i.e.*, 0's before 1's), and $b = 1$ to a negative one (*i.e.*, 1's before 0's). The cells $A(i, j, b)$ fulfilling $i < j$ and $i \equiv j + 1$ are of interest. These cells are filled using the following recursive rule, assuming $A(i, j, b) = 0$ when $i > j$:

$$A(i, j, b) = \begin{cases} 0 & j = i + 1 \text{ and } b \equiv i \\ f(w_i + w_j) & j = i + 1 \text{ and } b \equiv i + 1 \\ A(i + 1, j - 1, b) & j > i + 1 \text{ and } b \equiv i \end{cases} \tag{4}$$

If none of the above conditions holds, namely $j > i + 1$ and $b \equiv i + 1$, the rule is[4]:

[4]Notice that when $t = i$ and $k = j$ the equation assumes the cell $A(i, j, 1 - b)$ has a known value. Hence, when filling the matrix, $A(i, j, i \bmod 2)$ should be computed before $A(i, j, j \bmod 2)$.

Algorithm 4 `zerOneSort` (w)

```
1: calculate v(w)
2: for i = 0 to 2g + 1 do
3:    for j = 0 to 2g + 1 do
4:       for b = 0 to 1 do
5:          A(i, j, b) ← 0
6:       end for
7:    end for
8: end for
9: for j = 1 to 2g in steps of 1 do
10:   for i = j − 1 to 1 in steps of 2 do
11:      fill A(i, j, i mod 2) according to (4)
12:      fill A(i, j, j mod 2) according to (4) or (6)
13:   end for
14: end for
15: output A(1, 2g, 0)
```

$$A(i,j,b) = \min \left\{ \begin{array}{ll} A(i,t-1,b) + A(t,k,1-b) + & i \le t < k \le j, \\ A(k+1,j,b) + & : \quad t \equiv i, \text{ and} \\ f\left(\sum_{r=i,r\equiv t}^{k-1} w_r + \sum_{q=t+1,q\equiv k}^{j} w_q\right) & k \equiv j \end{array} \right\} \tag{5}$$

Calculating the sums in (5) when filling each cell increases the time complexity. To overcome this problem, we define a new representation of a 0/1 sequence, the *cumulative weighted* representation. Given a sequence $w = w_0, \ldots, w_{2g+1}$, define a new sequence $v = v(w) = v_0, \ldots, v_{2g+1}$, such that $v_i = \sum_{r=0,r\equiv i}^{i} w_r$. We can rewrite (5) using the cumulative representation as follows:

$$A(i,j,b) = \min \left\{ \begin{array}{ll} A(i,t-1,b) + A(t,k,1-b) + & i \le t < k \le j, \\ A(k+1,j,b) + & : \quad t \equiv i, \text{ and} \\ f\left(v_{k-1} - v_i + w_i + v_j - v_{t+1} + w_{t+1}\right) & k \equiv j \end{array} \right\} \tag{6}$$

The algorithm `zerOneSort` (Algorithm 4) uses (4) and (6) to calculate the minimum sorting cost of a 0/1 sequence.

Lemma 22. *The algorithm* `zerOneSort` *has time complexity in* $O(g^4)$ *and space complexity in* $O(g^2)$.

Proof. The space complexity of the matrix A is in $O(g^2)$, and so is the algorithm's space complexity.

Calculating the minimum in (6) requires $O(g^2)$ time. Therefore, the loop at line 9 dominates the time complexity and makes it in $O(g^4)$. □

Theorem 23. *The algorithm* `zerOneSort` *finds the minimum sorting cost of a given weighted sequence with time complexity in* $O(g^4)$ *and space complexity in* $O(g^2)$.

Proof. By Lemma 22, the time complexity of `zerOneSort` is in $O(g^4)$ and its space complexity is in $O(g^2)$. The optimality of the algorithm is guaranteed since the set $\mathscr{S}$ contains a minimum sorting series. □

4.3 A Faster Implementation

Using the additivity of the cost function, one can rewrite (5) to improve the algorithm's running time. Define a new matrix $B(x, y, d)$, for $x < y$, $x \equiv y + 1$, and $d \equiv x + 1$ as follows:

Algorithm 5 `fastZerOneSort` (w)

```
calculate v(w)
for i = 0 to 2g + 1 do
  for j = 0 to 2g + 1 do
    for b = 0 to 1 do
      A(i, j, b) ← 0
      B(i, j, b) ← 0
    end for
  end for
end for
for j = 1 to 2g in steps of 1 do
  for i = j − 1 to 1 in steps of 2 do
    fill A(i, j, i mod 2) according to (4)
    fill A(i, j, j mod 2) according to (4) or (8)
    fill B(i, j, j mod 2) according to (7)
  end for
end for
output A(1, 2g, 0)
```

$$B\left(x,y,d\right)=\min_{x<z\leq y,z\equiv y}\left\{\begin{array}{l}A\left(x,z,1-d\right)+A\left(z+1,y,d\right)+\\ f\left(\sum_{r=x,r\equiv x}^{z}w_r+\sum_{q=x+1,q\equiv y}^{y}w_q\right)\end{array}\right\} \tag{7}$$

The value of the cell $B(x, y, d)$ corresponds to the optimal cost of separating the subsequence $w_x, \ldots, w_y$ under the condition $c_x = 1$.

Using the matrix B, we can simplify (5) as follows:

$$A\left(i,j,b\right)=\min_{i\leq t\leq j,t\equiv i}\left\{\ A\left(i,t-1,b\right)+f\left(\sum_{r=i,r\equiv t}^{t-1}w_r\right)+B(t,j,b)\ \right\} \tag{8}$$

To avoid calculating the sum each time, (7) and (8) can be rewritten using $v(w)$. These equations reduce the running time of the algorithm by a factor of g, since each minimum can be calculated with time complexity in $O(g)$. The algorithm `fastZerOneSort` (Algorithm 5) implements this improvement.

Theorem 24. *The algorithm* `fastZerOneSort` *finds the minimum sorting cost of a given weighted sequence with time complexity in* $O(g^3)$ *and space complexity in* $O(g^2)$.

5 Properties of Optimal 0/1 Sorting when $\alpha = 1$

Here, we give proofs of the four properties introduced in Section 4.1. Throughout the proofs we use the weighted representation of 0/1 sequences. The proofs of the first three properties, in addition to a resulting exhaustive search algorithm are given first. The proof of the complex property is more elaborate and hence is given in its own subsection.

5.1 Trivial, Useless, and Cutting Reversals

The proofs of the trivial, useless, and cutting properties enable us to characterize the number of reversals in an optimal series. This characterization in turn enables us to introduce an efficient, though exponential,

algorithm for finding the optimal cost. In this section we prove the properties and give the resulting algorithm. The following notation is convenient for the proofs.

Given a 0/1 bit sequence $T = t_1, \ldots, t_n$ corresponding to a weighted sequence $w = w(T) = w_0, \ldots, w_{2g+1}$, map a block $t_i, \ldots, t_j$ in T to the pair (k, u) in w, such that $u = j - i + 1$ (the block's weight) and $k = i$ (the block's starting point). Let w_r be the weight in w to which k belongs, that is either $r = 0$ if $k \leq w_0$ or r fulfills $\sum_{q=0}^{r-1} w_q < k \leq \sum_{q=0}^{r} w_q$. We refer to u as a *subweight* of w_r. We drop the starting point k when there is no ambiguity in determining the location of the subweight u in w_r. If a reversal series ϱ does not cut the subsequence $t_i, \ldots, t_j$, define the reversal count of the subweight u by $N(u) = N(t_i, \ldots, t_j)$.

Lemma 25 (No Trivial Reversals). *A reversal series containing a trivial reversal cannot be optimal.*

Proof. The proof is by contradiction. Let $\varrho = \rho_1, \ldots, \rho_m$ be a minimum sorting series acting on a weighted sequence $w = w_0, \ldots, w_{2g+1}$, and containing a trivial reversal ρ_k. If ρ_k affects a subweight of w_0^{k-1} or $w_{|w^{k-1}|-1}^{k-1}$, exclude them from ρ_k and all subsequent reversals. The modified reversal series sorts w with a smaller cost. A contradiction. □

Lemma 25 enables us to pad a 0/1 sequence with a leading 0-block, and a closing 1-block.

Corollary 26. *Changing the values of the weights w_0 and w_{2g+1} does not affect the minimum sorting cost of a weighted sequence.*

Lemma 27 (No Useless Reversals). *A reversal series containing a useless reversal cannot be optimal.*

Proof. The proof is by contradiction. Let $\varrho = \rho_1, \ldots, \rho_m$ be a minimum sorting series acting on a weighted sequence $w = w_0, \ldots, w_{2g+1}$, and containing a useless reversal ρ_k. Denote the weights affected by ρ_k by $u_i, w_{i+1}, \ldots, w_{j-1}, u_j$ for $0 < u_i \leq w_i$ and $0 < u_j \leq w_j$. By Definition 13, since ρ_k is useless, we have $i \equiv j$. First, assume that $u_i \geq u_j$. Consider a modified reversal affecting the weights $u_i - u_j, w_{i+1}, \ldots, w_j$. This modified reversal has a smaller cost than ρ_k, while the 0/1 sequence that it produces is identical to the 0/1 sequence that ρ_k produces. Therefore, the original reversal sequence cannot be optimal. The remaining case $u_i < u_j$ is handled similarly. □

Lemma 28 (No Cutting Reversals). *A reversal series containing a cutting reversal cannot be optimal.*

Proof. The proof is by contradiction. Let $\varrho = \rho_1, \ldots, \rho_m$ be a minimum sorting series. By Lemma 27, the series ϱ contains no useless reversals. Consider the last cutting reversal ρ_k affecting weights $u_i, w_{i+1}, \ldots, w_{j-1}, u_j$ for $0 < u_i \leq w_i$ and $0 < u_j \leq w_j$. Assume that ρ_k cut the weight w_i, that is $u_i < w_i$ (the case $u_j < w_j$ is handled similarly). Notice that $N(u_i)$ and $N(w_i - u_i)$ are well defined under $\rho_k, \ldots, \rho_m$, since ρ_k is the last cutting reversal in the reversal series.

First, assume that $N(w_i - u_i) \leq N(u_i)$. Exclude u_i from the cutting reversal, and include it in all reversals in which $w_i - u_i$ takes part. Hence, the weight w_i moves as a unit through the reversals affecting $w_i - u_i$.

Since the cutting reversal is not useless, we have $i + 1 \equiv j$. Excluding u_i from ρ_k makes it a useless reversal. By Lemma 27 the modified series cannot be optimal.

The reversal counts of the elements of the 0/1 sequence do not increase by this modification. By (3), the modified reversal series has a cost less than or equal to the original one. Thus, the original series cannot be optimal.

The case $N(w_i - u_i) > N(u_i)$ is handled similarly. □

Corollary 29. *Let $w = w_0, \ldots, w_{2g+1}$ be a weighted sequence. An optimal sorting series contains exactly g reversals.*

Algorithm 6 `exhaustiveZerOneSort` (w)

```
1: if g = 0 then
2:    output 0
3: else
4:    opt ← ∞
5:    for i ← 1 to 2g − 1 in steps of 1 do
6:       for j ← i + 1 to 2g in steps of 2 do
7:          w ← w · ρ(i, j)
8:          tmp ← exhaustiveZerOneSort(w) + f (ρ(i, j))
9:          if tmp < opt then
10:            opt ← tmp
11:         end if
12:      end for
13:   end for
14:   output opt
15: end if
```

Proof. By definition, each reversal can reduce the length of a weighted sequence by at most two. Therefore, at least g reversals are required to sort w. By Lemmas 25, 27, and 28, an optimal sorting series contains no trivial, useless or cutting reversals. Thus, in an optimal solution, each reversal reduces the length of w exactly by two.

□

Corollary 29 implies that all optimal solutions are contained in the set of sorting series of length g. This set can be exhaustively searched for an optimal solution by a recursive algorithm. Given a 0/1 sequence $w = w_0, \ldots, w_{2g+1}$, choose a reversal $\rho(i, j)$, affecting the subsequence $w_i, \ldots, w_j$, such that $1 \leq i < j \leq 2g$ and $i + 1 \equiv j$. The reversal $\rho(i, j)$ is not trivial, useless, or cutting. We refer to such a reversal as a *legal* reversal. Perform $\rho(i, j)$ on w, sort $w \cdot \rho(i, j)$ recursively, and output the best result over all indices i and j. See Algorithm 6 (`exhaustiveZerOneSort`) for the pseudocode.

Lemma 30. *The time complexity of the algorithm* `exhaustiveZerOneSort` *is less than* $(g!)^2$.

Proof. There are less than g^2 legal reversals acting on a weighted 0/1 sequence of length $2g + 2$. Each legal reversal reduces the sequence' length by 2. Therefore, the number of nodes in the recursive tree is bounded by $g^2 \cdot (g-1)^2 \cdots 2^2 = (g!)^2$. □

Lemma 30 and Corollary 29 establish Theorem 31.

Theorem 31. *The algorithm* `exhaustiveZerOneSort` *sorts optimally a* 0/1 *sequence of length* $2g + 2$ *with time complexity smaller than* $(g!)^2$.

5.2 Complex Reversals

We show here that a minimum reversal series containing no complex reversals exists. This fact enables us to calculate the minimum cost in polynomial-time.

Let $w = w_0, \ldots, w_{2g+1}$ be a weighted sequence. Recall that $sg = w_i, \ldots, w_j$, or for short (i, j), is defined to be a segment of w. Let $sg_k = w_{i_k}, \ldots, w_{j_k}$ for $k \in \{1, 2\}$ be two segments of w. We say that sg_1 is a *sub-segment* of sg_2, or *contained* in sg_2, if $i_2 \leq i_1 < j_1 \leq j_2$. We say that sg_1 is *disjoint* to sg_2 if the intersection between the intervals (i_1, j_1) and (i_2, j_2) is empty.

Given two reversals $\rho = \rho(i_1, j_1)$ and $\eta = \eta(i_2, j_2)$, we say that ρ *contains* η or that ρ is *disjoint* to η if the segment (i_1, j_1) contains or is disjoint to the segment (i_2, j_2), respectively.

Given a reversal $\rho = \rho(i, j)$ acting on a weighted sequence $w = w_0, \ldots, w_{2g+1}$, we say that ρ *unifies* w_i with w_{j+1}, and w_j with w_{i-1} if $i \equiv j + 1$. The reversal ρ is called a *unifying* reversal. In addition, we say that the weight w'_{i-1} of $w' = w \cdot \rho = w'_1, \ldots, w'_{2g-2}$ *contains* the weights w_{i-1} and w_j of w. Similarly we say that the weight w'_{j-1} of w' contains the weights w_{j+1} and w_i of w.

A reversal series $\varrho = \rho_1, \ldots, \rho_m$ acting on w *separates* a segment $sg = (i, j)$ of w, if ϱ unifies all the 0- and 1-weights of sg. The reversal series ϱ separates the segment sg *positively* or with *positive polarity*, if ϱ unifies all weights of the segment into two weights $w'_{i'}$ and $w'_{j'}$ of $w' = w \cdot \rho_1 \cdots \rho_m$, such that $i' \equiv 0$, $j' \equiv 1$, and $i' < j'$. The reversal series ϱ separates the segment sg *negatively* or with *negative polarity* if $i' > j'$. If a reversal series ϱ separates positively the segment $(0, 2g + 1)$ of w, then ϱ sorts w.

The polarity of a two-block segment $sg = (i, i+1)$ is *positive* if $i \equiv 0$ and *negative* otherwise. The polarity of a simple reversal ρ equals the polarity of the two-block segment that ρ affects.

Proposition 32. *Let ϱ be a reversal series that separates a segment sg of a 0/1 sequence w. Then, the reversal series ϱ separates each sub-segment sg' of sg. Furthermore, the restriction of ϱ to sg' separates sg'.*

Proof. Since sg' is contained in sg, if sg' is not separated, sg cannot be separated. Furthermore, if the restriction of ϱ to sg' does not separate sg', ϱ cannot separate sg'. □

In the sorting process, the essence of a reversal is not directly connected to the coordinates (i, j) that define the reversal, but is designated by the weights that the reversal affects. This essence plays a crucial rule when defining *commutative* reversals.

Given a reversal ρ acting on w and a reversal η acting on $w \cdot \rho$, we say that η *commutes* with ρ if two reversals η' and ρ' exist such that $w \cdot \rho \cdot \eta = w \cdot \eta' \cdot \rho'$, where η' and ρ' affect the same weights of w that η and ρ affected respectively (in the following we drop the primes).

The following proposition characterizes commutative reversals.

Proposition 33. *Let w be a weighted series, ρ be a reversal acting on it, and η be a reversal acting on $w \cdot \rho$. The three following conditions are equivalent:*

1. *The reversal η commutes with ρ.*
2. *The weights that η affects in $w \cdot \rho$ are contiguous in w.*
3. *The reversals ρ and η are either contained one in the other or disjoint to each other.*

Let ϱ be a reversal series separating a segment sg of w. We refer to the maximal[5] segment that contains sg and is separated by ϱ as msg.

Given a weighted sequence w and a minimum sorting series $\varrho = \rho_1, \ldots, \rho_m$, let ρ_j be the complex reversal having the greatest index . The reversal ρ_j affects a segment s_2 of w^{j-1}. Let k be the smallest index such that $\rho_j, \ldots, \rho_k$ separates s_2[6], and let msg be the maximal segment of w^{j-1} containing s_2 that $\rho_j, \ldots, \rho_k$ separates. Denote the segment to the left of s_2 in msg by s_1, and the segment to the right of s_2 in msg by s_3[7].

$$w^{j-1} = w_0^{j-1}, \ldots, w_q^{j-1}, \overbrace{\overbrace{\ldots, w^{j-1}_{q+|s_1|}}^{s_1}, \underbrace{\ldots, w^{j-1}_{q+|s_1|+|s_2|}}_{\rho_j}}^{s_2}, \overbrace{\ldots, w^{j-1}_{q+|s_1|+|s_2|+|s_3|}}^{s_3}, \ldots \quad .$$

[5] With respect to the containment partial order on segments of w.

[6] The subseries $\rho_j, \ldots, \rho_m$ separates w^{j-1}. By Proposition 32 it separates s_2. Hence, k is well defined.

[7] The segments s_1 and s_3 might be empty.

The following lemmas characterize the (simple) reversals performed after ρ_j.

Lemma 34. *Let w, ρ_j, s_2, k, msg, s_1, and s_3 be as above. Consider a reversal ρ_r for $j < r \leq k$. If ρ_r affects at least one mixed weight, i.e., a weight that contains weights from s_1 and s_2 or s_2 and s_3, then all the weights that ρ_r affects become part of mixed weights.*

Proof. By induction on r. Base case: consider the sequence w^j. Since ρ_j is not trivial, useless, or cutting, it unifies $w^{j-1}_{q+|s_1|}$ with $w^{j-1}_{q+|s_1|+|s_2|}$ producing $w^j_{q+|s_1|}$ and $w^{j-1}_{q+|s_1|+1}$ with $w^{j-1}_{q+|s_1|+|s_2|+1}$ producing $w^j_{q+|s_1|+|s_2|-1}$. These weights are the only mixed weights in w^j. If ρ_{j+1} is to affect mixed weights, it must affect one of them and a neighboring weight.

To see that all the weights that ρ_{j+1} affects become part of mixed weights, assume that ρ_{j+1} affects weights $w^j_{q+|s_1|}$ and $w^j_{q+|s_1|+1}$ (the other cases are handled similarly). This act causes the mixed weight $w^j_{q+|s_1|}$ to unite with $w^j_{q+|s_1|+2}$ resulting in a bigger mixed weight, while the weight $w^j_{q+|s_1|+1}$ from s_2 unites with the weight $w^j_{q+|s_1|-1}$ from s_1 resulting in the formation of a new mixed weight on this edge of s_2[8].

The induction step is established similarly. □

Notice that since the subseries $\rho_{j+1}, \ldots, \rho_k$ contains no complex reversals, the mixed weights remain at the edges of s_2. This fact is helpful for proving the following lemma.

Lemma 35. *Let w, ρ_j, s_2, k, msg, s_1, and s_3 be as in Lemma 34 and let ρ_r for $j < r \leq k$ be the smallest index reversal that does not affect mixed weights. Then ρ_r commutes with all the reversals ρ_q for $j \leq q < r$.*

Proof. By Lemma 34, since all the reversals ρ_q for $j < q < r$ affect weights that turn to be part of mixed weights, and since ρ_r affects no mixed weights, it is disjoint to ρ_q for $j < q < r$. By Proposition 33, ρ_r commutes with ρ_q for $j < q < r$.

Since the reversal ρ_r does not affect mixed weights, it is either contained in or disjoint to s_2. In either case, by Proposition 33, ρ_r commutes with ρ_j. □

Corollary 36. *Let w, ρ_j, s_2, k, msg, s_1, and s_3 be as in Lemma 35. We can rearrange the reversal series so that all the reversals ρ_r for $j < r \leq k$ affect mixed weights.*

Proof. By induction on the number of reversals ρ_r for $j < r \leq k$ that do not affect mixed weights and Lemma 35. □

By Corollary 36, the reversals ρ_r for $j < r \leq k$ affect mixed weights. In addition, since the reversals ρ_r for $j < r \leq m$ are good, they fulfill Lemma 20. These two properties are helpful for proving that there exists an optimal series containing no complex reversals.

Lemma 37. *There exists an optimal reversal series containing no complex reversals.*

Proof. By induction on the length of the weighted sequence. Base case: let w be a sequence of length four. By Corollary 29, an optimal series sorting w must be of length one. A single complex reversal, however, cannot sort w. Therefore, all optimal sorting series do not contain a complex reversal. Induction step: assume the claim holds for all sequences of length smaller than or equal to a, where $a \geq 4$. We need to prove the claim for $a + 2$. Let w be a weighted sequence of length $a + 2$, and assume the claim does not hold. Consider an optimal sorting series $\varrho = \rho_1, \ldots, \rho_m$ having the minimum number of complex reversals, and let ρ_j be the complex reversal with the greatest index. Let s_2, k, msg, s_1, and s_3 be as in Corollary 36 and let ρ_r for $j < r \leq k$ be the outcome of the corollary, *i.e.*, all these reversals affect mixed weights. The reversal ρ_j remains a complex reversal. Otherwise, we get a minimum sorting series having a smaller number of complex reversals. Since the reversals ρ_r for $j < r \leq m$ are good, by Lemma 20, there exist indices $i' \equiv 0$

[8] There can be two different mixed weights at each edge of s_2: a 0-mixed weight and a 1-mixed weight.

and $p' \equiv 1$ in w^j, such that $i' > p'$ and $c_{i'} = c_{p'} = 1$. Notice that each of the weights $w^j_{i'}$ and $w^j_{p'}$ could be present as a unit in w^{j-1} or could result from the unification of two weights of w^{j-1}. In the former case, pick the weight in w^{j-1} corresponding to $w^j_{i'}$ or $w^j_{p'}$, respectively. In the latter, notice that the unification occurs between a weight from s_2 and a weight from either s_1 or s_3. Choose the the weight not belonging to s_2. Denote the indices of the chosen weights in w^{j-1} by i and p, respectively; see Figure 4.

$$\begin{array}{lccccccccccc} w^{j-1} = & 2 & 8 & 4 & [15 & \overbrace{2}^{i} & \overbrace{4}^{p} & 7] & 5 \\ w^j = & 2 & 8 & 11 & \underbrace{4}_{p'} & \underbrace{2}_{i'} & 20 \end{array}$$

(a)

$$\begin{array}{lccccccccccc} w^{j-1} = & 2 & 8 & \overbrace{4}^{p} & [15 & \overbrace{2}^{i} & 4 & 7] & 5 \\ w^j = & 2 & 8 & \underbrace{11}_{p'} & 4 & \underbrace{2}_{i'} & 20 \end{array}$$

(b)

Figure 4: The process of choosing the weights w^{j-1}_i and w^{j-1}_p. (a) If the weights corresponding to i' and p' in w^j are not mixed, *i.e.*, were not unified by ρ_j, choose the corresponding blocks in w^{j-1}. (b) If one of the chosen weights in w^j is unified (p'), choose the corresponding weight in w^{j-1} that was not affected by the reversal.

According to the position of the indices i and p we divide the analysis into three cases:

1. The indices i and p do not belong to s_2. The segment s_2 can be located between i and p, can be located to the left of p, or can be located to the right of i; see Figure. Here, we handle the first case; the other two cases are handled similarly. Denote the segment between i and p by sg. Notice that $\rho_j, \ldots, \rho_m$ separates sg. Let k' be the smallest index such that $\rho_j, \ldots, \rho_{k'}$ separates sg. Since s_2 is a subsegment of sg, by Proposition 32 it is separated as well. Therefore, $k \leq k'$ and msg is a subsegment of sg. Without loss of generality, all the reversals ρ_r for $j < r \leq k'$ affect sg. Since sg is of length less than $a+2$, and by the induction assumption, we can replace $\rho_j, \ldots, \rho_{k'}$ by a minimum cost reversal series having no complex reversals[9]. This modification yields a minimum sorting series of w having a smaller number of complex reversals than ϱ has. A contradiction.

$$\begin{array}{lccccccccccccccc} & \leftarrow & s_1 & \rightarrow & \longleftarrow & s_2 & \longrightarrow & \leftarrow & s_3 & \rightarrow \\ \rho_j & 2 & 5 & \overbrace{3}^{p} & [2 & 7 & 6 & 4] & \overbrace{5}^{i} & 10 & 12 \end{array}$$

(a)

$$\begin{array}{lccccccccccccccc} & \leftarrow & s_1 & \rightarrow & \longleftarrow & s_2 & \longrightarrow & \leftarrow & s_3 & \rightarrow \\ \rho_j & 2 & 5 & 3 & [2 & 7 & 6 & 4] & \overbrace{5}^{p} & \overbrace{10}^{i} & 12 \end{array}$$

(b)

$$\begin{array}{lccccccccccccccc} & \leftarrow & s_1 & \rightarrow & \longleftarrow & s_2 & \longrightarrow & \leftarrow & s_3 & \rightarrow \\ \rho_j & 2 & \overbrace{5}^{p} & \overbrace{3}^{i} & [2 & 7 & 6 & 4] & 5 & 10 & 12 \end{array}$$

(c)

Figure 5: The different options in case 1 of the proof of Lemma 37. In all three cases the complex reversal can be eliminated by the induction assumption (see text). (a) The segment s_2 is located between i and p. (b) The segment s_2 is located to the left of p. (c) The segment s_2 is locate to the right of i.

2. The indices i and p belong to s_2. In this case s_2 is separated only when w^j is sorted, that is $msg = w^j$ and $k = m$. Furthermore, we have $|i - p| = 1$, since otherwise the reversals acting on the weights between i and p during the separation of s_2 commute with ρ_j. Notice that s_1 and s_3 are positively separated under ρ_r for $j < r < m$, because ρ_j does not affect them and ρ_m affects only the 1's of s_1 and the 0's of s_3. Moreover, the reversals ρ_r for $j < r \leq m$ separates s_2 negatively, since ρ_j affects all

[9] In case of negative separation, flip the segment sg to get a positive separation, replace $\rho_j, \ldots, \rho_{k'}$ with a minimum sorting series having no complex reversals, and reflip the segment sg.

the weights in s_2 and hence changes only the orientation of the separation. Skip ρ_j and perform the reversals ρ_r for $j < r < m$ restricted to each of s_1, s_2, and s_3, and perform ρ_m restricted to s_2. Thus, the segments s_1 and s_3 get positively separated, while s_2 gets negatively separated. Therefore, the bit sequence corresponding to the weighted sequence gets the form: $T = \overbrace{0^+1^+}^{s_1}\ \overbrace{1^+0^+}^{s_2}\ \overbrace{0^+1^+}^{s_3}$. Here, 0^+ (1^+) corresponds to a maximal contiguous block of 0's (1's). Since ρ_j is not performed in the modified series, all elements in s_2 have reversal counts at least smaller by one than the original series. We say that s_2's elements have one *available reversal*. Similarly, since ρ_m is not performed on s_1 and s_3, the 1-block of s_1 and the 0-block of s_3 have one available reversal. Perform an additional reversal on T (reversal indicated by brackets $[,]$): $\overbrace{0^+[1^+}^{s_1}\ \overbrace{1^+0^+}^{s_2}\ \overbrace{0^+]1^+}^{s_3}$. This additional reversal sorts the sequence. The reversal counts of the elements in the modified series are not greater than the reversal counts in the original one. By (3) the modified series cost is not greater than the original one. However, the modified series has less complex reversals than the original one. A contradiction. Figure 6 gives an example illustrating the modification.

Counts	0	2	1	2	2	2	2	1	2	0
		$\leftarrow s_1 \rightarrow$		$\longleftarrow$		s_2	$\longrightarrow$		$\leftarrow s_3 \rightarrow$	
					$\overbrace{\ \ }^{i}$	$\overbrace{\ \ }^{p}$				
ρ_j	2	5	3	[2	7	6	4]	5	10	12
				$\overbrace{\ \ }^{p'}$	$\overbrace{\ \ }^{i'}$					
ρ_{j+1}	2	[5	7]	6	7	7	10	12		
ρ_{j+2}	9	11	7	[7	10]	12				
ρ_m	9	[11	17]	19						
sorted	26	30								

(a)

Counts	0	2	1	2	2	2	2	1	2	0
		$\longleftarrow s_1 \longrightarrow$		$\longleftarrow$		s_2	$\longrightarrow$		$\leftarrow s_3 \rightarrow$	
					$\overbrace{\ \ }^{i}$	$\overbrace{\ \ }^{p}$				
skip ρ_j	2	5	3	2	7	6	4	5	10	12
					$\overbrace{\ \ }^{i}$	$\overbrace{\ \ }^{p}$				
$\rho_{j+1}\|s_2, \rho_{j+1}\|s_1$	2	[5	3]	2	7	6	[4]	5	10	12
$\rho_{j+2}\|s_2, \rho_{j+2}\|s_3$	5	5 + [2]	7	6	4	[5	10]	12		
$\rho_m\|s_2$	5	7	[7	6]	14	17				
add. reversal	5	[13	21]	17						
sorted	26	30								

(b)

Figure 6: An example illustrating the modification described in case 2 of the proof of Lemma 37. The reversal counts for each scenario are given at the top. (a) The original reversal series starting with the last complex reversal ρ_j. (b) The modification suggested in case 2 of the proof of Lemma 37. Restrictions of reversals from (a) are given in parallel. The additional reversal is indicated as "add. reversal" (see case 2 in the proof of Lemma 37). Notice that the modification results in the same reversal counts as the original series. However, the modification does not contain any complex reversals.

3. One of the indices is contained in s_2, while the other is not. Here, again, s_2 is separated only when w is sorted, that is $msg = w$ and $k = m$. We handle the case in which i is contained in s_2 and p in s_1. To handle the other case, flip the sequence and rename the 0's and the 1's to get the former case.

   Since $k = m$, all the reversals performed after ρ_j affect mixed weights. Therefore, we must have $p = 1$, or else the reversals affecting the weights to the left of p cannot affect mixed weights.

   Perform the reversals ρ_r for $j < r < m$ restricted to each of s_1, s_2, and s_3 in w^{j-1}. The bit sequence gets the following form (assuming $w_0 = 0$).

$$T = \overbrace{\underbrace{1^+}_{\text{Block } p=1}\ 1^+0^+}^{s_1}\ \overbrace{1^+0^+\ \underbrace{0^+}_{\text{Block } i}\ 0^+1^+}^{s_2}\ \overbrace{0^+1^+}^{s_3}\ .$$

   Checking the available reversal counts shows that the same technique used before cannot be applied here. Therefore, a more careful analysis is needed. Denote the subblocks of s_1 by $s_{1,1}$ and $s_{1,2}$

respectively. Denote the left 1-subblock of s_2 by $s_{2,1}$, the right 1-subblock by $s_{2,3}$, and the 0-subblock by $s_{2,2}$. Thus, we have:

$$T = \overbrace{\underbrace{\underbrace{1^+}_{\text{Block } p}\ 1^+}_{s_{1,1}}\ \underbrace{0^+}_{s_{1,2}}}^{s_1}\ \overbrace{\underbrace{1^+}_{s_{2,1}}\ \underbrace{0^+\ \underbrace{0^+}_{\text{Block } i}\ 0^+}_{s_{2,2}}\ \underbrace{1^+}_{s_{2,3}}}^{s_2}\ \overbrace{0^+1^+}^{s_3} \ .$$

Consider the sequence w^{j-1} and the neighborhood of the weight i in it. Denote the closest 1-block in $s_{2,1}$ to i by $s_{2,1,0}$ and denote the next closest 1-block by $s_{2,1,1}$. Denote the closest 1-block in $s_{2,3}$ to i by $s_{2,3,0}$ and denote the next closest 1-block by $s_{2,3,1}$:

$$T(w^{j-1}) = \cdots \overbrace{1^+}^{s_{2,1,1}}\ 0^+\ \overbrace{1^+}^{s_{2,1,0}}\ \overbrace{0^+}^{i}\ \overbrace{1^+}^{s_{2,3,0}}\ 0^+\ \overbrace{1^+}^{s_{2,3,1}} \cdots \ .$$

In the general case, the blocks $s_{2,1,1}$ and $s_{2,3,1}$ might not exist. If $s_{2,3,1}$ does not exist, we define $N(s_{2,3,1}) = 0$. Similarly, if $s_{2,1,1}$ does not exist we define $N(s_{2,1,1}) = 0$. Since in this case i cannot be a block at the edge of s_2 and since the reveral ρ_j is complex, $s_{2,1,0}$ and $s_{2,3,0}$ must always exist.

By Lemma 34 the block $s_{2,1,0}$ is the last 1-block to join $s_{2,1}$ in the original reversal series, and the block $s_{2,1,1}$ is its predecessor (see also Figure 7a). A similar claim holds for $s_{2,3,0}$, $s_{2,3,1}$, and $s_{2,3}$. Denote by $s_{2,1}/s_{2,1,0}$ all blocks belonging to $s_{2,1}$ except $s_{2,1,0}$. Define $s_{2,3}/s_{2,3,0}$ similarly. Recall that $N(s_{2,3})$ is defined to be the number of reversals in which $s_{2,3}$ takes part as a contiguous block (for example, ρ_j is not counted in it, but ρ_m is). Here, we start counting the reversal counts from ρ_{j+1}, *i.e.*, we omit ρ_j from the counts. In the following, we suggest a modified reversal series based on a comparison between the reversal counts of $s_{2,1}$, $s_{2,1}/s_{2,1,0}$, $s_{2,3}$, and $s_{2,3}/s_{2,3,0}$ in the original reversal series. The intuition behind the strategy of the modification is simple: If $N(s_{2,1}) > N(s_{2,3})$, then we can sort the sequence by letting $s_{2,1}$ participate in the reversals affecting $s_{2,3}$ without violating its reversal counts. On the other hand, the block $s_{2,3}$ is "smuggled" to its original position relative to i and participates in the reversals that affected it in the original series. In all modifications the complex reversal ρ_j is not performed.

(a) If $N(s_{2,1}) \geq N(s_{2,3})$.

i. If $N(s_{2,1}) \geq N(s_{2,3}/s_{2,3,0})$.
Perform the reversals restricted to s_2 and s_3. Perform reversals affecting $s_{2,1}$ restricted to s_3, but not to s_2. Perform the reversals restricted to s_1 till reaching the reversal unifying $s_{2,3,1}$ with $s_{1,1}$. Let the block $s_{2,1}$ participate in each reversal that $s_{2,3,1}$ takes part in, till reaching the reversal unifying $s_{2,3,0}$ with $s_{1,1}$.
By now, a part of $s_{1,2}$ is unified with $s_{2,2}$. We refer to this block as $s'_{2,2}$. Perform the unifying reversal restricted only to s_2 to unify $s_{2,3}$ and perform a reversal affecting $s_{2,3}$ and $s'_{2,2}$. Let $s_{2,3}$ take part in all the remaining reversals as in the original series.
To show that the modified series's cost is not greater than the original one, consider the reversal count of each of the sequence's blocks. Notice that $N(s_{2,3,0}) \leq N(s_{2,3}) + 1$ since $s_{2,3,0}$ might be affected by a reversal that unifies it with $s_{2,3}/s_{2,3,0}$ and after that the whole block moves together as a unit. Similarly one gets $N(s_{2,3,1}) \leq N(s_{2,3}/s_{2,3,0}) + 1$. From the assumptions for this case, we get $N(s_{2,3,1}) \leq N(s_{2,1}) + 1$. Since ρ_j is not performed, all elements of s_2 have one available reversal. Thus, the number of reversals that $s_{2,1}$, $s_{2,2}$, and $s_{2,3}$ can take part in equals their original reversal counts plus 1, which implies that $s_{2,1}$ can replace $s_{2,3,1}$ while $s_{2,3}$ can replace $s_{2,3,0}$ without violating the available reversal counts.

Original series	Modified series
Reversals affecting s_2 and s_3	Restrict them to s_2 and to s_3. If a reversal affects $s_{2,1}$, skip performing the restriction to s_2.
Reversals affecting s_1 and s_2 but not unifying $s_{2,3,1}$ with $s_{1,1}$.	Restrict them to s_1 and to s_2.
The reversal unifying $s_{2,3,1}$ with $s_{1,1}$.	Restrict it to s_2 (to unify $s_{2,3,1}$ with $s_{2,3}$) and let $s_{2,1}$ participate instead of $s_{2,3,1}$ in the reversal restricted to s_1 and in all comming reversals.
Reversals affecting s_1 and s_2 but not unifying $s_{2,3,0}$ with $s_{1,1}$.	Restrict them to s_1 and to s_2, with the modification regarding $s_{2,1}$.
The reversal unifying $s_{2,3,0}$ with $s_{1,1}$.	Restrict the reversal to s_2 (to unify $s_{2,3,0}$ with $s_{2,3}$) and perform a reversal affecting $s_{2,3}$ and $s'_{2,2}$ to unify $s_{2,3}$ with $s_{1,1}$.
Reversals affecting s_1 and s_2.	Restrict them to s_1 and to s_2, with the modification regarding $s_{2,1}$. In addition, let $s_{2,3}$ participate in all reversals as in the original series.
ρ_m	Perform ρ_m with the modification regarding $s_{2,1}$.

Table 2: A comparison between the original series and the modified one in case 3(a)i of the proof of Lemma 37. The original series can be divided into three subseries (excluding ρ_j): reversals affecting s_1 and s_2, reversals affecting s_2 and s_3, and reversals affecting s_1, s_2, and s_3 (only ρ_m). The modifications in the table are given in this order. Notice that the first two subseries commute, since reversals beloging to them are disjoint (they affect different sides of i).

Notice that $s_{2,2}$ takes part in an additional reversal (relative to the original series) when unifying $s_{2,3}$ with $s_{1,1}$. That reversal is balanced by skipping ρ_j. It is easy to see that the reversal counts of the rest of the elements in the sequence are not violated. Therefore, by (3) the modified series's cost is not greater than the original one. However, the modified series has a smaller number of complex reversals. A contradiction. Table 2 summarizes the changes discussed above and Figure 7 gives an example illustrating the modification.

ii. If $N(s_{2,1}) < N(s_{2,3}/s_{2,3,0})$.
In this case, $s_{2,1}$ cannot replace $s_{2,3,1}$, but it can replace $s_{2,3,0}$. Furthermore, $s_{2,3}/s_{2,3,0}$ can replace $s_{2,1,0}$. These replacements are done as follows: perform the reversals restricted to s_2 except the reversal unifying $s_{2,3,0}$ with $s_{2,3}$. Instead, unify $s_{2,3,0}$ with $s_{2,1}$ by performing a reversal affecting $s_{2,3,0}$ and the parts already formed of $s_{2,2}$ (the block containing weight i). Refer to the unified block as $s_{2,1} \cup s_{2,3,0}$. Now, perform the reversals restricted to s_1 and let the reversals affecting $s_{2,3,0}$ affect $s_{2,1} \cup s_{2,3,0}$ instead. Perform the reversals restricted to s_3 and let the reversals affecting $s_{2,1,0}$ affect $s_{2,3}/s_{2,3,0}$ instead.
A contradiction is established similarly to the previous case.

(b) If $N(s_{2,1}) < N(s_{2,3})$.
This case is symmetrical to the former and can be handled similarly.

Since we reached a contradiction in all cases, we proved the claim. □

Notice that in case 3. of the proof, if we omit the leading 1-weight of w^{j-1} and the last reversal ρ_m, the problem becomes equivalent to finding a minimum reversal series with no complex reversals transforming a 0/1 sequence to the form $1^+0^+1^+$. The same proof, with minor changes, applies to the latter claim, which is useful for proving the complex property when dealing with circular 0/1 sequences (see [28] for more details).

Corollary 38. *There exists a minimum reversal series containing no complex reversals transforming a* 0/1 *sequence to the form* $1^+0^+1^+$.

Counts	0	1	3	4	4	3	2	3	3	3	4	3	2	1	2	0
		$\leftarrow$	s_1	$\rightarrow$	$\longleftarrow$			s_2			$\longrightarrow$	$\leftarrow$		s_3		$\rightarrow$
ρ_j	0	$\overbrace{5}^{p}$	3	$[\overbrace{5}^{s_{2,1,1}}$	3	$\overbrace{2}^{s_{2,1,0}}$	$\overbrace{7}^{i}$	$\overbrace{6}^{s_{2,3,0}}$	4	$\overbrace{5}^{s_{2,3,1}}$	10]	1	4	7	9	0
ρ_{j+1}	0	$\overbrace{5}^{p}$	[13	5]	4	6	$\overbrace{7}^{i}$	$\overbrace{2}^{s_{2,1,0}}$	3	6	4	7	9	0		
ρ_{j+2}	0	$\overbrace{10}^{p}$	17	6	$\overbrace{7}^{i}$	$\overbrace{2}^{s_{2,1,0}}$	[3	6]	4	7	9	0				
ρ_{j+3}	0	$\overbrace{10}^{p}$	[17	$6]\overbrace{7}^{i}$	$\overbrace{8}^{s'_{2,1}}$	7	7	9	0							
ρ_{j+4}	0	$\overbrace{16}^{p}$	$\overbrace{24}^{i}$	$[\overbrace{8}^{s'_{2,1}}$	7]	7	9	0								
ρ_{j+5}	0	$\overbrace{16}^{p}$	$\overbrace{31}^{i}$	[15	9]	0										
ρ_m	0	$[\overbrace{16}^{p}$	$\overbrace{40}^{i}]$	15												
sorted	40	31														

(a)

Counts	0	1	3	3	4	2	2	3	3	3	4	3	2	1	2	0
		$\leftarrow$	s_1	$\rightarrow$	$\longleftarrow$				s_2			$\longrightarrow$	$\leftarrow$	s_3		$\rightarrow$
skip ρ_j	0	$\overbrace{5}^{p}$	3	$\overbrace{5}^{s_{2,1,1}}$	3	$\overbrace{2}^{s_{2,1,0}}$	$\overbrace{7}^{i}$	$\overbrace{6}^{s_{2,3,0}}$	4	$\overbrace{5}^{s_{2,3,1}}$	10	1	4	7	9	0
$\rho_{j+2}\|s_2, \rho_{j+2}\|s_3$	0	$\overbrace{5}^{p}$	3	$[\overbrace{5}^{s_{2,1,1}}$	3]	$\overbrace{2}^{s_{2,1,0}}$	$\overbrace{7}^{i}$	$\overbrace{6}^{s_{2,3,0}}$	4	$\overbrace{5}^{s_{2,3,1}}$	10	[1]	4	7	9	0
~~$\rho_{j+3}\|s_2$~~, $\rho_{j+4}\|s_3$	0	$\overbrace{5}^{p}$	6	$\overbrace{7}^{s_{2,1}}$	$\overbrace{7}^{i}$	$\overbrace{6}^{s_{2,3,0}}$	4	$\overbrace{5}^{s_{2,3,1}}$	10		[1	4]	7	9	0	
~~$\rho_{j+4}\|s_2$~~, $\rho_{j+5}\|s_3$	0	$\overbrace{5}^{p}$	6	$\overbrace{7}^{s_{2,1}}$	$\overbrace{7}^{i}$	$\overbrace{6}^{s_{2,3,0}}$	4	$\overbrace{5}^{s_{2,3,1}}$	10 + 4		[8	9]	0			
ρ_{j+1} modified	0	$\overbrace{5}^{p}$	[6	$\overbrace{7}^{s_{2,1}}]$	$\overbrace{7}^{i}$	$\overbrace{6}^{s_{2,3,0}}$	4	$[\overbrace{5}^{s_{2,3,1}}$	10] + 13		8					
$\rho_{j+3}\|s_2$	0	$\overbrace{12}^{p}$	$\overbrace{13}^{i}$	$[\overbrace{6}^{s_{2,3,0}}$	14]	$\overbrace{5}^{s_{2,3,1}}$	13	8								
reverse $s_{2,3} + s'_{2,2}$	0	$\overbrace{12}^{p}$	$[\overbrace{27}^{i,s'_{2,2}}$	$\overbrace{11}^{s_{2,3}}]$	13	8										
ρ_m modified	0	[23	40]	8												
sorted	40	31														

(b)

Figure 7: An example illustrating the modification suggested in case 3(a)i of the proof of Lemma 37. The reversal counts for each scenario are given at the top. Note that the cost of the series in (b) is less than the cost of the series in (b). (a) The original reversal series starting from the complex reversal ρ_j. Notice that $N(s_{2,1}) = 2$, $N(s_{2,3}) = 1$, and $N(s_{2,3}/s_{2,3,0}) = N(s_{2,3,1}) = 2$ (excluding ρ_j). Thus, the assumptions of case 3(a)i hold. $s'_{2,1}$ denotes a weight that includes $s_{2,1}$. (b) The modification suggested in case 3(a)i of the proof of Lemma 37. The reversals restricted to s_2 and s_3 are performed first. Restrictions affecting $s_{2,1}$ are skipped (striked like ~~this~~). The reversal ρ_{j+1} is modified to unify $s_{2,1}$ (instead of $s_{2,3,1}$) with $s_{1,1}$ (denoted here by p) and its restriction to s_2 is performed as well. The reversal ρ_{j+3} (unifying $s_{2,3,0}$ with $s_{1,1}$) is performed restricted to s_2 to unify $s_{2,3}$, then $s_{2,3}$ and $s'_{2,2}$ are reversed to unify the former with $s_{1,1}$ ("reverse $s_{2,3} + s'_{2,2}$"). From this point, $s_{2,1}$ and $s_{2,3}$ take part in all the reversals that affected $s_{2,3}$ in the original series, *e.g.*, in a modification of ρ_m.

6 Approximation Algorithms When $\alpha \geq 0$

We now consider the problem of approximating the optimal cost to sort a given sequence. To achieve good approximation ratios, we need different algorithms for the different ranges of α. In contrast, recall that Section 3 presents a single divide-and-conquer algorithm that achieves optimal or nearly optimal sorting bounds for all $0 \leq \alpha < 2$.

6.1 Approximation Algorithm When $1 < \alpha < 2$

The sorting algorithm from Section 3.1 for $1 < \alpha < 2$ does not deliver a good approximation ratio. To see why, consider the permutation $\pi = n, 1, 2, 3, \ldots, n-1$. The optimal solution ($n-1$ reversals of length 2) has cost $\Theta(n)$, whereas the sorting algorithm has cost $\Theta(n^\alpha)$. The moral is that an approximation algorithm for $\alpha > 1$ is different from one for $\alpha = 1$, where sorting all out-of-order regions yields an $O(\lg^2 n)$ approximation [24].

We begin by explaining some properties of the cost function $f(\ell) = \ell^\alpha$ when $1 < \alpha < 2$.

Observation 39. *Let T_1 and T_2 be disjoint subsequences of bit sequence T (i.e., the subsequences may interleave but have no common elements). Then for any reversal ρ in T and cost function $f(\ell) = \ell^\alpha$ $(1 < \alpha < 2)$, the cost of the reversal ρ is at least the cost of the reversal restricted to T_1 plus the cost of the reversal restricted to T_2, that is,*

$$f(\rho) \geq f(\rho\,|T_1) + f(\rho\,|T_2).$$

Proof. Let the length of the reversal T be ℓ, the length of $\rho\,|T_1$ be ℓ_1, and the length of $\rho\,|T_2$ be ℓ_2. Because T_1 and T_2 are disjoint, we know that $\ell \geq \ell_1 + \ell_2$. Since $\alpha > 1$,

$$f(\rho) = \ell^\alpha \geq \ell_1^\alpha + \ell_2^\alpha = f(\rho\,|T_1) + f(\rho\,|T_2).$$

□

Corollary 40. *Let T_1 and T_2 be disjoint subsequences of sequence T. Then the optimal cost to sort T is at least the sum of the optimal costs to sort T_1 and T_2 for cost function $f(\ell) = \ell^\alpha$ $(1 < \alpha < 2)$.*

Proof. Assume that $\varrho = \rho_1, \ldots, \rho_m$ is the optimal reversal series to sort T. Then, $\varrho|T_1 = \rho_1|T_1, \ldots, \rho_m|T_1$ and $\varrho|T_2 = \rho_1|T_2, \ldots, \rho_m|T_2$ are reversal series to sort disjoint subsequences T_1 and T_2, which are greater than the optimal for T_1 and T_2. From Observation 39, we have

$$f(\varrho) = \sum_{i=1}^{m} f(\rho_i) \geq \sum_{i=1}^{m} (f(\rho_i|T_1) + f(\rho_i|T_2)) = f(\varrho|T_1) + f(\varrho|T_2).$$

□

BSBR: $O(1)$ Approximation Algorithm When $1 < \alpha < 2$

We now give a divide-and-conquer approximation algorithm, `kBasedDC` (Algorithm 7), for sorting 0/1 sequences. We later use this algorithm as a subroutine for sorting permutations.

Suppose the sequence has k 0's and $n-k$ 1's. Split the sequence at position k. This split means that the sequence has k elements to its left and $n-k$ elements to its right. Sort both left and right subsequences recursively. This recursive step results in a sequence of the form $0 \cdots 01110001 \cdots 1$; see Figure 8. To complete the sorting, perform one more reversal of the first block of 1's and the second block of 0's.

Algorithm 7 `kBasedDC` (T)

1: $k \leftarrow \#_0(T)$
2: $c_1 \leftarrow$ `kBasedDC` $(t_1, \ldots, t_k)$
3: $c_2 \leftarrow$ `kBasedDC` $(t_{k+1}, \ldots, t_n)$
4: $i \leftarrow \#_1(t_1, \ldots, t_k)$
5: $j \leftarrow \#_0(t_{k+1}, \ldots, t_n)$
6: $T \leftarrow T \cdot \rho(k - i + 1, k + j)$
7: return $c_1 + c_2 + f(i + j)$

Theorem 41. *The algorithm* `kBasedDC` *(Algorithm 7) is an $O(1)$-approximation algorithm for sorting 0/1 sequences when $1 < \alpha < 2$.*

To prove Theorem 41, we first give a lower bound using a potential-function argument. Then, we prove that the sorting cost of `kBasedDC` is within a constant factor of the initial potential value.

We now define a potential function $W(T)$ for any 0/1 sequence T.

Definition 42. For a 0/1 sequence T of length n and any integer $1 \le i \le n$, define the number of *wrong-sided elements* $w(i, T)$ for position i to be the number of extra 1's in the first i elements in T plus the number of extra 0's in the last $n - i$ elements in T when compared to the sorted sequence. We define the potential function $W(T)$ as follows:

$$W(T) = \sum_{i=1}^{n} w(i, T)^{\alpha - 1}.$$

Lemma 43. *A reversal ρ of length ℓ on bit sequence T decreases the value of the potential function $W(T)$ by at most ℓ^α, that is, $W(T) - W(T \cdot \rho) \le \ell^\alpha$.*

Proof. For the reversal ρ and any integer $1 \le i \le n$, the value of $w(i, T)$ is changed by the reversal ρ only if the position i is inside ρ, which means that at most ℓ of the $w(i, T)$ terms change. Because the length of the reversal is ℓ, the value of $w(i, T)$ changes by at most ℓ, that is, $w(i, T) - w(i, T \cdot \rho) \le \ell$. Because $0 < \alpha - 1 < 1$, we have $w(i, T)^{\alpha-1} - w(i, T \cdot \rho)^{\alpha-1} \le \ell^{\alpha-1}$. Thus,

$$W(T) - W(T \cdot \rho) = \sum_{i \in \rho} \left[w(i, T)^{\alpha-1} - w(i, T \cdot \rho)^{\alpha-1} \right] \le \ell \cdot \ell^{\alpha-1} = \ell^\alpha.$$

□

We obtain the following corollary:

Corollary 44. *The potential function $W(T)$ is a lower bound on the cost to sort the sequence T by reversals when $1 < \alpha < 2$.*

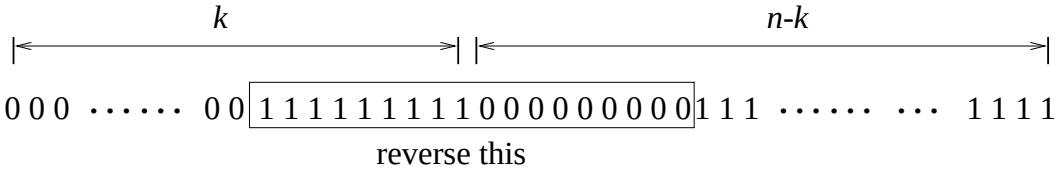

Figure 8: An illustration of step 6 in Algorithm `kBasedDC` (Algorithm 7).

Proof. The cost for a reversal of length ℓ is $f(\ell) = \ell^\alpha$ and the value of $W(T)$ is 0 for the sorted sequence. From Lemma 43, we know that each reversal decreases the potential value by at most ℓ^α. Thus, $W(T)$ is a lower bound. □

Now we prove that Algorithm 7 sorts using cost $O(W(T))$, which establishes an $O(1)$-approximation ratio. We first prove a lemma about the number $w(i,T)$ of wrong-sided elements.

Lemma 45. *If T is a bit sequence of length n and i is an integer $1 \le i \le n$, and we add a 0 or 1 to either the right or the left end of T to create a new sequence T', the value of $w(i,T)$ increases monotonically, i.e., $w(i,T') \ge w(i,T)$.*

Proof. If a 0 is added to the right end, then the extra number of 1's on the left side of i and the extra number of 0's on the right side of i only increases. If a 1 is added to the right end, then the extra number of 1's on the left side of i and the extra number of 0's on the right side of i do not change. The argument is symmetric when a 0 or 1 is added to the left side of the sequence. Thus, $w(i,T)$ only increases when we extend the sequence at either end. □

Thus, we have the following corollary:

Corollary 46. *If T is a bit sequence, T_L is the subsequence of the left k elements, and T_R is the subsequence of the right $n-k$ elements, then for any $1 \le i \le n$,*

$$w(i,T) \ge \begin{cases} w(i,T_L), & 1 \le i \le k, \\ w(i-k,T_R), & k < i \le n. \end{cases}$$

Proof. The number of wrong-sided elements $w(i,T_L)$ is increased to $w(i,T)$ by adding T_R of length $n-k$ to T_L's right end. The number of wrong-sided elements $w(i-k,T_R)$ is increased to $w(i,T)$ by adding T_L of length k to T_R's left end. □

Lemma 47. *The algorithm `kBasedDC` (Algorithm 7) sorts any sequence T with cost $O(W(T))$.*

Proof. Recall that there are k 0's, so the lengths of T_L and T_R are k and $n-k$, respectively. Define cost $\mathcal{C}(T)$ to be the cost of this algorithm to sort sequence T.

In the last "conquer" step of `kBasedDC`, we reverse $w(k,T)$ wrong-sided elements with respect to position k for a cost of $w(k,T)^\alpha$. Thus, we have the following recurrence for $\mathcal{C}(T)$:

$$\mathcal{C}(T) = \mathcal{C}(T_L) + \mathcal{C}(T_R) + w(k,T)^\alpha.$$

For future analysis, we now want to prove that

$$W(T) - W(T_L) - W(T_R) \ge c\, w(k,T)^\alpha,$$

for some constant c. To do so, we define

$$\Delta(i) = \begin{cases} [w(i,T)]^{\alpha-1} - [w(i,T_L)]^{\alpha-1}, & i \le k, \\ [w(i,T)]^{\alpha-1} - [w(i-k,T_R)]^{\alpha-1}, & i > k. \end{cases}$$

From Corollary 46, we know that $\Delta(i) \ge 0$. Therefore,

$$\begin{aligned} W(T) - W(T_L) - W(T_R) &= \sum_{i=1}^{n} \Delta(i) \\ &\ge \sum_{i=k-w(k,T)/4}^{k+w(k,T)/4} \Delta(i). \end{aligned}$$

Because shifting the position i in the sequence to the left or right by 1 can change the number of wrong-sided elements by at most 2, for any $1 \le j \le w(k,T)/4$, $w(k-j,T) \ge w(k,T) - 2j$ and $w(k+j,T) \ge w(k,T) - 2j$.

Notice $w(k,T_L) = 0$ and $w(0,T_R) = 0$. For the same reason as above we know that for any $1 \le j \le w(k,T)/4$, $w(k-j,T_L) \le w(k,T_L) + 2j = 2j$ and $w(j,T_R) \le w(0,T_R) + 2j = 2j$.

Thus, since $\left[w(k,T) - 2j\right]^{\alpha-1} \ge (2j)^{\alpha-1}$ when $0 \le j \le w(k,T)/4$, we have

$$\begin{aligned}
W(T) - W(T_L) - W(T_R) &\ge \sum_{i=k-w(k,T)/4}^{k+w(k,T)/4} \Delta(i) \\
&\ge \sum_{j=0}^{w(k,T)/4} \left\{ \left[w(k,T) - 2j\right]^{\alpha-1} - (2j)^{\alpha-1} \right\} \\
&\ge \sum_{j=0}^{w(k,T)/8} \left\{ \left[w(k,T) - 2j\right]^{\alpha-1} - (2j)^{\alpha-1} \right\} \\
&\ge \frac{w(k,T)}{8} \left[(3/4)^{\alpha-1} - (1/4)^{\alpha-1}\right] w(k,T)^{\alpha-1} \\
&= \frac{1}{8} \left[(3/4)^{\alpha-1} - (1/4)^{\alpha-1}\right] w(k,T)^{\alpha}.
\end{aligned}$$

By induction on the length of T, we prove that $W(T) \ge c\,\mathcal{C}(T)$, where

$$c = \min\left\{ \frac{1}{8}\left[(3/4)^{\alpha-1} - (1/4)^{\alpha-1}\right], \frac{1}{2} \right\}.$$

Base case: if $|T| = 2$, then we have $W(T) = 2^{\alpha-1}$ and $\mathcal{C}(T) = 2^{\alpha}$, and the inequality holds. Induction step: assume the inequality holds for all sequences of length smaller than or equal to n. We need to prove the inequality for sequences T of length $n+1$. Since T is of length $n+1$, the subsequences T_L and T_R are of length smaller than or equal to n. Hence, by the induction step:

$$W(T) \ge W(T_L) + W(T_R) + c w(k,T)^{\alpha} \ge c[\mathcal{C}(T_L) + \mathcal{C}(T_R) + w(k,T)^{\alpha}] = c\,\mathcal{C}(T).$$

□

PSBR: $O(\lg n)$ Approximation Algorithm When $1 < \alpha < 2$

We give an approximation algorithm for sorting a permutation π, which is a surprising enhancement of the sorting Algorithm `PermutationSort_DivideConquer` (Algorithm 3) from Section 3.1. We add one intermediate step: after we divide the sequence π into two halves about the median and sort them as a 0/1 sequence using the Algorithm `kBasedDC` (Algorithm 7) with reversal series $\varrho = \rho_1 \cdots \rho_m$, we sort each half to return the elements to the same order as in π. Let π_L and π_R be the left and right halves after performing reversal series ϱ. Then, $\pi_L \cdot \varrho^{Tran}|_{\pi_L}$ and $\pi_R \cdot \varrho^{Tran}|_{\pi_R}$ will return the elements in each half to their original order, here $\varrho^{Tran} = \rho_m \cdots \rho_1$ is the reversed order of ϱ. After this step we recursively sort each half. At first glance, this modification seems to increase the complexity, but, in fact the complexity is reduced enough to approximate the optimal sorting cost to within a logarithmic factor.

Algorithm 8 `reorderReversalSort` has the following performance:

Theorem 48. *The algorithm `reorderReversalSort` (Algorithm 8) is an $O(\lg n)$ approximation algorithm when $1 < \alpha < 2$.*

Proof. Let $\mathrm{opt}(\pi)$ be the optimal cost to sort permutation π. The cost for Step 2 is at most $O(\mathrm{opt}(\pi))$ by Lemma 47. The cost for Step 3 is at most the cost of Step 2, and hence is at most $O(\mathrm{opt}(\pi))$. This

Algorithm 8 `reorderReversalSort` (π)

1: $T \leftarrow \text{permTo01}(\pi)$
2: `kBasedDC` (T), with reversal series ϱ
3: $\pi'_L \leftarrow \pi_L \cdot \varrho^{Tran}|_{\pi_L}$ and $\pi'_R \leftarrow \pi_R \cdot \varrho^{Tran}|_{\pi_R}$
4: `reorderReversalSort` (π'_L)
5: `reorderReversalSort` (π'_R)

follows from Observation 39, and because the inverse of a reversal is itself, and we are just doing the inverse restricted permutation on left and right subsequences. We also know from Observation 39 that $\text{opt}(\pi'_L) + \text{opt}(\pi'_R) \leq \text{opt}(\pi)$ because π'_L and π'_R are disjoint subsequences of the original sequence π. Thus, the cost of Algorithm 8 is

$$\mathcal{C}(\pi) = \mathcal{C}(\pi'_L) + \mathcal{C}(\pi'_R) + \text{cost of Steps 2 and 3,}$$

and with a simple induction we establish the $O(\lg n)$ approximation. □

6.2 PSBR: $O(\lg n)$ Approximation Algorithm When $\alpha = 1$

The Algorithm `ReorderSortLinearF` (Algorithm 9) for the case of $\alpha = 1$ is modified from Algorithm `reorderReversalSort` (Algorithm 8) by using the optimal 0/1 sorting algorithm designed for $\alpha = 1$. Using Algorithm 4 (`zerOneSort`) as a subroutine in Step 2 of Algorithm 9 (`ReorderSortLinearF`) guarantees a logarithmic approximation ratio for $\alpha = 1$. The proof uses similar ideas to those in Theorem 48.

Algorithm 9 `ReorderSortLinearF` (π)

1: $T \leftarrow \text{permTo01}(\pi)$
2: `zerOneSort` (T) (Algorithm 4), with reversal series ϱ
3: $\pi'_L \leftarrow \pi_L \cdot \varrho^{Tran}|_{\pi_L}$ and $\pi'_R \leftarrow \pi_R \cdot \varrho^{Tran}|_{\pi_R}$
4: `ReorderSortLinearF` (π'_L)
5: `ReorderSortLinearF` (π'_R)

Theorem 49. *The Algorithm `ReorderSortLinearF` (Algorithm 9) is an $O(\lg n)$ approximation algorithm when $\alpha = 1$.*

Proof. In each recursive round, the Algorithm `zerOneSort` (Algorithm 4) costs less than the optimal permutation sort, since sorting the permutation implies sorting the induced 0/1 sequence. The result follows because we perform $\Theta(\lg n)$ levels of recursive of Algorithm 4, `zerOneSort`. □

6.3 Approximation Algorithms When $0 \leq \alpha < 1$

We first give an $O(\lg n)$-approximation algorithm for sorting 0/1 sequences. A 0/1 sequence can be viewed as composed of zero blocks (0's) and one blocks (1's). Without loss of generality, suppose the sequence is in this form: $1^{w_1}0^{w_2}\cdots 1^{w_{2g-1}}0^{w_{2g}}$. By symmetry, all other cases can be reduced to this case. We have the following lower bound:

Lemma 50. *A lower bound on* $\text{opt}(T)$ *to sort a sequence* $T = 1^{w_1}0^{w_2}\cdots 1^{w_{2g-1}}0^{w_{2g}}$ *by reversals when* $0 \leq \alpha < 1$ *is*

$$V(T) = \frac{1}{2}\sum_{i=1}^{2g} |w_i|^{\alpha}.$$

Proof. We prove by induction on j that if a solution uses exactly j reversals, then the cost is at least $V(T)$. When $j = 1$, one reversal sorts the blocks. So the original sequence must be $1^{w_1}0^{w_2}$. The cost is $(|w_1|+|w_2|)^\alpha$, which is greater than $(|w_1|^\alpha + |w_2|^\alpha)/2$.

Suppose for all integers less than j, the claim holds. Then for $j+1$, assume the first reversal ρ of length ℓ in the optimal solution changes the sequence T to T', where T' has a solution with j reversals. We know from the inductive assumption assumption that the cost $\text{opt}(T') \geq V(T')$. Because the optimal cost to sort T is the cost to sort T' plus the cost of the reversal ρ,

$$\text{opt}(T) = \text{opt}(T') + \ell^\alpha \geq V(T') + \ell^\alpha.$$

We show $V(T') + \ell^\alpha \geq V(T)$ in the following, thus proving that $V(T)$ is a lower bound.

Since $0 \leq \alpha < 1$, for any positive integers x and y, we have $x^\alpha + y^\alpha \geq (x+y)^\alpha$. Then we know, the decrease of $V(T)$ by one reversal can only be caused by two blocks (or even sub-blocks generated by reversal ρ's edge splitting a block) merging into each other. Let two blocks B_1 (out of reversal ρ) of length a and B_2 (in reversal ρ, might be a sub-block from ρ) of length b merge each other at one side of reversal ρ, then the decrease of $V(T)$ is $[a^\alpha + b^\alpha - (a+b)^\alpha]/2$, which is less than $b^\alpha/2 \leq \ell^\alpha/2$. Note that a reversal can *at most merge two blocks at both sides*, we know that the whole decrease of $V(T)$ is at most $\ell^\alpha/2 + \ell^\alpha/2 = \ell^\alpha$, which means

$$V(T) - V(T') \leq \ell^\alpha.$$

□

BSBR: $O(\lg n)$ Approximation Algorithm When $0 \leq \alpha < 1$

The approximation algorithm is based on divide-and-conquer: Map each block of 0's or 1's to a single element 0 or 1 in a new sequence T', ignoring the block of 0's at the leftmost position and the block of 1's at the rightmost position if they exist. Use Algorithm 1 (`ZerOneSort_DivideConquer`) from Section 3 to determine the reversals to sort sequence T'. Map back each element in T' onto a block in T, and map each reversal back according to the same mapping. Algorithm 10 (`BlockMapping`) implements this mapping.

Algorithm 10 `BlockMapping` (T', ρ)

1: $l \leftarrow$ left end of ρ
2: $r \leftarrow$ right end of ρ
3: $l' \leftarrow \sum_{j=1}^{l-1} T'_j.weight + 1$
4: $r' \leftarrow \sum_{j=1}^{r} T'_j.weight$
5: return $\rho'(l', r')$

Algorithm 11 `BlockDC` (T)

1: scan T to find $(w_1, w_2, \ldots, w_{2g})$ such that $T = 1^{w_1}0^{w_2}\cdots 1^{w_{2g-1}}0^{w_{2g}}$
2: $T' \leftarrow 1010\cdots10$ and $T'_i \sup .weight = w_i$
3: `ZerOneSort_DivideConquer` (T'), with reversal series $\varrho = \rho_1 \cdots \rho_m$
4: **for** $j = 1$ to m **do**
5: $\quad \rho_T \leftarrow BlockMapping(T' \cdot \rho_1 \cdots \rho_{j-1}, \rho_j)$
6: $\quad T \leftarrow T \cdot \rho_T$
7: **end for**

Thus, we just perform the standard divide-and-conquer algorithm from Section 3, but on the 0/1 blocks. The performance guarantees are based on the following structural lemma:

Lemma 51. *In algorithm* `BlockDC` *(Algorithm 11) each element takes part in at most* $\lg n$ *reversals.*

Proof. For each recursive step, an element appears in at most one reversal. There are $\lg n$ recursive steps, and thus each element appears in at most $\lg n$ reversals. □

The above algorithm performs as follows:

Theorem 52. *The algorithm* `BlockDC` *(Algorithm 11) is an* $O(\lg n)$*-approximation algorithm when* $0 \leq \alpha < 1$.

Proof. Suppose reversal ρ of length ℓ contains blocks $w_p, w_{p+1}, \ldots, w_q$. Because when $0 \leq \alpha < 1$, $\ell^\alpha \leq \sum_{i=p}^{q} w_i^\alpha$. Thus, the total cost is at most the sum of cost for reversing each block times the number of reversals containing the block, which is at most $\lg n$ times the total cost for reversing the blocks, which is $2V(T)$. Because $V(T)$ is a lower bound on opt, we obtain an $O(\lg n)$-approximation. □

BSBR: $O(1)$ Approximation Algorithm When $0 \leq \alpha < 1$

We obtain a constant approximation by improving the splitting as follows: If there is any 0/1 block of size at least $n/3$, perform a reversal of length at most n to move this block to the edge of the sequence (a 0 block moves to the front, and a 1 block moves to the back). Then remove this block from the sequence T and sort the rest of sequence T' recursively. If there are no blocks of 0's or 1's of size at least $n/3$, then there exists a block edge at a distance at least $n/3$ from both ends. Split the whole sequence T at this edge to form left and right subsequences T_L and T_R. Sort T_L and T_R recursively, then perform a reversal of length at most $n = |T_L| + |T_R|$, and the sequence T is sorted.

Algorithm 12 `ImprovedDC` (T)

```
scan T to find (w_1, w_2, ..., w_{2g}) such that T = 1^{w_1} 0^{w_2} ... 1^{w_{2g-1}} 0^{w_{2g}}
if w_i ≥ n/3 then
    if i is odd then
        T = 1^{w_1} ... 0^{w_{i-1}} 0^{w_{2g}} 1^{w_{2g-1}} ... 0^{w_{i+1}}
    else
        T = 1^{w_{i-1}} 0^{w_{i-2}} ... 1^{w_1} 1^{w_{i+1}} ... 0^{w_{2g}}
    end if
end if
find left half T_L and right half T_R such that |T_L|, |T_R| ≥ n/3
ImprovedDC (T_L)
ImprovedDC (T_R)
i ← 1 + #_0(T_L)
j ← |T_L| + #_0(T_R)
perform one more reversal ρ(i, j) to finish sorting
```

Theorem 53. *The algorithm* `ImprovedDC` *(Algorithm 12) is an* $O(1)$ *approximation algorithm for sorting* 0/1 *sequences for* $\alpha = 1$.

Now we prove that this algorithm will sort any 0/1 sequence T with cost at most $O(V(T))$. Before we begin the main proof, we need some definitions and lemmas:

Definition 54. Define constant t is

$$t = \frac{1}{(1/3)^\alpha + (2/3)^\alpha} \ (< 1).$$

Define functions $\delta(T)$ and $\beta(T)$ of sequence T recursively as follows:

- For a subsequence T' composed only of 0's or only of 1's with length b, $\delta(T') = \beta(T') = b^\alpha$.
- If there is any 0/1 block B of size $w \geq n/3$, we let $\delta(T) = t\delta(T') + w^\alpha$ and $\beta(T) = \beta(T') + w^\alpha$, where T' is the subsequence of T with the block B removed.
- If there is no 0/1 block of size at least $n/3$, then we can find a block B whose distances from one end to both ends of the sequence T are bigger than $n/3$. We split the sequence T at this end to two subsequence: left one T_L and right one T_R. Let $\delta(T) = t[\delta(T_L) + \delta(T_R)]$ and $\beta(T) = \beta(T_L) + \beta(T_R) + \delta(T)$.

Lemma 55. *For $\frac{n}{3} \leq x \leq n$, we have that $g(x) \triangleq t(n-x)^\alpha + x^\alpha - n^\alpha \geq 0$.*

Proof. First, because $g''(x) = \alpha(\alpha-1)[t(n-x)^{\alpha-2} + x^{\alpha-2}] < 0$, $g(x)$ is upper concave. If we check the value of $g(x)$ at the boundary $x = n/3$, we have that

$$g(x) = \left[\frac{1}{(1/3)^\alpha + (2/3)^\alpha}(2/3)^\alpha + (1/3)^\alpha\right] n^\alpha - n^\alpha \geq \left[\frac{(1/3)^\alpha + (2/3)^\alpha}{(1/3)^\alpha + (2/3)^\alpha} - 1\right] n^\alpha = 0.$$

At the boundary $x = n$, $g(x) = 0$.

Thus, we have $g(x) \geq 0$ for all $n/3 \leq x \leq n$. At the same time, if $n/3 \leq x \leq 2n/3$, then we get the stronger inequality using the same method of proof:

$$t(n-x)^\alpha + tx^\alpha - n^\alpha \geq 0. \tag{9}$$

□

Lemma 56. *For any 0/1 block sequence $T = 1^{w_1}0^{w_2}\cdots 1^{w_{2g-1}}0^{w_{2g}}$, we have*

$$\beta(T) \leq \frac{1}{1-t}\sum_{i=1}^{2g} w_i^\alpha = \frac{2}{1-t}V(T).$$

Proof. If we write down $\beta(T)$ in terms of w_i^α, we observe that it is the sum of $t^j \cdot w_i^\alpha$, and each (i,j) pair could appear at most once. Thus $\beta(T)$ is bounded above by the sum

$$\sum_{i=1}^{2g}\sum_{j=0}^{\infty} t^j w_i^\alpha = \frac{1}{1-t}\sum_{i=1}^{2g} w_i^\alpha.$$

□

Lemma 57. *For any sequence T of length n, we have $\delta(T) \geq n^\alpha$.*

Proof. We prove this by induction on r, the number of blocks in T. The base case is that the sequence is just one 0/1 block. Then lemma holds trivially.

We now assume that the claim holds when the number of blocks is less than r. Then for $r+1$ blocks, there are two cases as follows:

If there is any 0/1 block B of size w at least $n/3$, from Lemma 55, we have

$$\delta(T) = t\delta(T') + w^\alpha \geq t(n-w)^\alpha + w^\alpha \geq n^\alpha.$$

If there is no 0/1 block of size at least $n/3$, then from (9), we have

$$\delta(T) = t[\delta(T_L) + \delta(T_R)] \geq t(|T_L|^\alpha + |T_R|^\alpha) \geq n^\alpha.$$

□

Define $\mathcal{C}(T)$ to be the cost of sorting 0/1 sequence T using Algorithm 12 (`ImprovedDC`). The following theorem will also establish Theorem 53.

Theorem 58. *Let T be a 0/1 sequence of length n, composed of 0/1 blocks $B_1, B_2, \ldots, B_r$ of sizes $w_1, w_2, \ldots, w_r$. The sorting cost of Algorithm 12 when applied to T is less than $3^\alpha \beta(T) = O(V(T))$.*

Proof. The proof is by induction on the number of blocks r.

The base case is $r = 1$, *i.e.*, the sequence is a single block. We can sort it with no cost, which is less than $\beta(T)$.

For inductive step, assume that the assumption holds for all number of blocks at most r. Then for $r+1$ blocks, there are two cases: one is that there is some block B of size w at least $n/3$; the other is that all blocks have size less than $n/3$.

In the case one, we do a reversal of length at most n to move this block B to the edge of sequence T. Let T' be the subsequence of T after removing the block B, then $\beta(T) = \beta(T') + w^\alpha$. There are at most r blocks in the sequence T', which means $\mathcal{C}(T') \leq 3^\alpha \beta(T')$. Now we have

$$\mathcal{C}(T) \leq n^\alpha + \mathcal{C}(T') \leq (3w)^\alpha + 3^\alpha \beta(T') = 3^\alpha \beta(T).$$

In the case two, if T_L and T_R are the left and right subsequences in the algorithm, then $\beta(T) = \beta(T_L) + \beta(T_R) + \delta(T)$. Because we can sort the sequence T by sorting T_L and T_R recursively and one more reversal with length at most n across the median, then we have that

$$\mathcal{C}(T) \leq \mathcal{C}(T_L) + \mathcal{C}(T_R) + n^\alpha \leq 3^\alpha \beta(T_L) + 3^\alpha \beta(T_R) + \delta(T) \leq 3^\alpha \beta(T).$$

□

6.4 Approximation and Exact Algorithms for Large α

We now give exact and approximate algorithms for sorting by reversals for large α. We show that when $\alpha \geq 2$, bubble-sort is optimal for sorting 0/1 sequences, and when $\alpha \geq 3$ it is also optimal for permutations.

BSBR When $\alpha \geq 2$

We consider first the sub-class of algorithms using only reversals of length 2.

Lemma 59. *When $\alpha \geq 2$, bubble sort is optimal among all algorithms using only reversals of length 2.*

Proof. Let $T = t_1, \ldots, t_n$ be a 0/1 sequence, and consider reversal series composed of length-2 reversals. We say that two elements t_i and t_j are *in correct order* if $t_i \leq t_j,\ i < j$. We define an order function $X(i,j)$ for element pair t_i, t_j to be

$$X(i,j) = \begin{cases} 0, & t_i \leq t_j \text{ and } i < j, \\ 1, & \text{otherwise.} \end{cases}$$

For any sequence T, we define the potential function to be the number of pairs of elements in reverse order:

$$P(T) = \sum_{1 \leq i < j \leq n} X(i,j).$$

Each reversal decreases the potential value by at most 1. Thus, it requires at least $P(T)$ reversals to sort T and sorting cost at least $2^\alpha P(T)$. Since each reversal in bubble-sort decreases this number by exactly 1, bubble sort is optimal over algorithms using only length-2 reversals. □

We now show that if $\alpha \geq 2$, it never makes sense to perform reversals greater than 2.

Theorem 60. *Bubble-sort sorts* 0/1 *sequences optimally when* $\alpha \geq 2$.

Proof. We prove that there are no reversals of length greater than 2 in the optimal sorting sequence. For the sake of contradiction, suppose there is at least one reversal of length $\ell \geq 3$ in the optimal sorting sequence. The cost of this reversal is ℓ^α. Suppose that there are k 1's and $\ell - k$ 0's in this reversal. Replace this reversal in the solution by length-2 reversals with bubble-sort. For each 0 element, bubble-sort requires k length-2 reversals (since each 0 element needs to be flipped with k 1's). In total, bubble-sort requires $k(\ell-k)$ length-2 reversals for the $\ell - k$ 0's. Therefore, its total cost is $k(\ell - k)2^\alpha$. Observe that

$$k(\ell - k)2^\alpha \leq \frac{\ell}{2}\frac{\ell}{2}2^\alpha \leq 2^{\alpha-2}\ell^2 < \ell^\alpha.$$

Thus, we can replace any reversal of length at least 3 with length-2 reversals with lower cost, meaning that no reversals of length greater than 2 appear in the optimal sorting sequence.

From Lemma 59, bubble-sort is optimal among all sorting solutions with length-2 reversals, establishing the theorem. □

Sorting Permutations for $\alpha \geq 2$ and $\alpha \geq 3$

We now show how bubble-sort performs on permutations for large α. For $\alpha \geq 2$, we have the following approximation bound:

Theorem 61. *Bubble-sort is a* 2*-approximation algorithm for sorting permutations in the case* $\alpha \geq 2$.

Proof. We show that given an optimal sorting sequence of cost $\text{opt}(\pi)$, there is a length-2-reversal solution of cost at most $2\text{opt}(\pi)$. For each reversal in the optimal sequence, replace it by a sequence of reversals of length 2 with bubble-sort. A reversal of length ℓ is replaced by $\ell(\ell-1)/2$ length-2 reversals using bubble-sort. Therefore the total cost of the new length-2 reversals sorting sequence is at most

$$2^\alpha \cdot \frac{\ell(\ell-1)}{2} \leq 2^{\alpha-1}\ell^2 \leq 2\ell^{\alpha-2}\ell^2 = 2\ell^\alpha.$$

□

Once $\alpha \geq 3$, bubble-sort becomes optimal. Thus, Caprara's hardness proof [10] for $\alpha = 0$ does not extend automatically to all cost functions.

Theorem 62. *Bubble-sort optimally sorts permutations when* $\alpha \geq 3$.

Proof. We show that reversals of length greater than 2 can be replaced by length-2 reversals while decreasing the sorting cost. Consider an optimal sorting sequence, and suppose it has a reversal of length $\ell \geq 3$ with sorting cost of ℓ^α. Replace this reversal with reversals of length-2 using bubble-sort. The cost of bubble-sort in this step is at most $2^{\alpha-1}\ell(\ell-1)$.

If $\ell = 3$, then

$$\frac{3(3-1)}{2} \cdot 2^\alpha \leq 3 \cdot 2^\alpha < 3^\alpha.$$

If $\ell \geq 4$, we have:

$$2^\alpha \frac{\ell(\ell-1)}{2} < 2^{\alpha-1}\ell^2 \leq \ell^{\alpha-2}\ell^2 = \ell^\alpha.$$

Thus, we can replace any long reversal by length-2 reversals at a lower cost, meaning that any reversal of length great than 2 cannot occur in the optimal sorting sequence. The theorem follows from Lemma 59. □

7 Conclusions

In this paper we studied the problem of sorting by length-weighted reversals. There are many previous papers on sorting by reversals, and most earlier papers assume that all reversals have the same cost. The motivation for much work on sorting by reversals, including this paper, comes from comparative genomics, and in this field, it has been shown to be reasonable to weight the cost of a reversal by its length ℓ. This paper presents the first comprehensive analysis of sorting by length-weighted reversals, for a wide range of cost functions $f(\ell) = \ell^\alpha$.

The length-weighted problem introduces richness and variety to sorting by reversals for both sorting 0/1 sequences and permutations. In our algorithms, a main strategy was to use 0/1-sorting as the principal workhorses for sorting permutations. We found that the 0/1 problem becomes appealing once reversals have weights. In particular, if $\alpha = 0$, then the problem is straightforward: any reversal reduces the number of 0/1 blocks by at most two. On the other hand, for $\alpha > 0$, the problem is sensitive to the length and the order of the reversals; a special structure was needed to show that the problem remains solvable in polynomial time for $\alpha = 1$. The problem is still open for other values of $\alpha \in (0, 1) \cup (1, 2)$.

We found one universal algorithm for sorting all "interesting" values of α, *i.e.*, $\alpha \in (0, 2)$, although we needed different proofs to show that its worst-case performance was tight or near tight for different ranges of α. However, for both 0/1 sequences and permutations, we needed different algorithms to approximate the optimal sorting cost for $\alpha \in (0, 1)$ and $\alpha \in (1, 2)$. It is still an open question to find a nontrivial approximation algorithm for sorting permutations for $0 < \alpha < 1$.

Many questions remain about applying these models and techniques to comparative genomics. For example, it is an open question which cost function is most biologically relevant. It remains to be shown whether a single cost function is appropriate for all biological applications or whether different functions should be used in different situations.

References

[1] Y. Ajana, J. Lefebvre, E. Tillier, and N. El-Mabrouk. Exploring the set of all minimal sequences of reversals — an application to test the replication-directed reversal hypothesis. In *Second Int. Workshop on Algorithms in Bioinformatics (WABI'02)*, volume 2452, pages 300–315. Springer-Verlag Lecture Notes in Computer Science, 2002.

[2] V. Bafna and P. Pevzner. Genome rearrangements and sorting by reversals. In *Proc. 34th IEEE Symp. of the Foundations of Computer Science (FOCS)*, pages 148–157. IEEE Computer Society Press, 1994.

[3] V. Bafna and P. Pevzner. Sorting by reversals: Genome rearrangements in plant organelles and evolutionary history of X chromosome. *Molecular Biology and Evolution*, 1994.

[4] V. Bafna and P. Pevzner. Sorting permutations by transpositions. In *Proceedings of the 6th Annual Symposium on Discrete Algorithms (SODA)*, pages 614–623, 1995.

[5] V. Bafna and P. Pevzner. Genome rearrangements and sorting by reversals. *SIAM J. Computing*, 25:272–289, 1996.

[6] M. A. Bender, D. Ge, S. He, H. Hu, R. Y. Pinter, S. Skiena, and F. Swidan. Improved bounds on sorting with length-weighted reversals. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 919–928, 2004.

[7] A. Bergeron. A very elementary presentation of the hannenhalli-pevzner theory. In *Proc. 12th Symp. Combinatorial Pattern Matching (CPM)*, volume 2089, pages 106–117. Springer-Verlag Lecture Notes in Computer Science, 2001.

[8] P. Berman, S. Hannenhalli, and M. Karpinski. 1.375–approximation algorithm for sorting by reversals. In *10th European Symposium on Algorithms (ESA)*, pages 200–210. Springer-Verlag Lecture Notes in Computer Science, 2002.

[9] M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric genome rearrangement. *Gene*, 172:GC:11–17, 1996.

[10] A. Caprara. Sorting by reversals is difficult. In *Proc. First Annual International Conference on Computational Molecular Biology (RECOMB'97)*, pages 75–83, 1997.

[11] D. A. Christie. A 3/2-approximation algorithm for sorting by reversals. In *Proceedings of the 9th Annual Symposium on Discrete Algorithms (SODA)*, pages 244–252, 1998.

[12] M. Davisson. X-linked genetic homologies between mouse and man. *Genomics*, 1:213–227, 1987.

[13] T. Dobzhansky and A.H.Sturtevant. Inversions in the chromosomes of *drosophila pseudoobscura*. *Genetics*, 23:28–64, 1938.

[14] S. Hannenhalli, C. Chappey, E. Koonin, and P. Pevzner. Scenarios for genome rearrangements: Herpesvirus evolution as a test case. In *Proc. of 3rd Intl. Conference on Bioinformatics and Complex Genome Analysis*, 1994.

[15] S. Hannenhalli and P. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problems. In *Proc. 36th IEEE Symp. of the Foundations of Computer Science (FOCS)*, pages 581–592. IEEE Computer Society Press, 1995.

[16] S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *J. ACM*, 46:1–27, 1999.

[17] H. Kaplan, R. Shamir, and R. E. Tarjan. Faster and simpler algorithm for sorting signed permutations by reversals. In *Proceedings of the 8th Annual Symposium on Discrete Algorithms (SODA)*, pages 344–351, 1997. Also in Proc. RECOMB 97, page 163.

[18] J. Kececioglu and R. Ravi. Physical mapping of chromosomes using unique probes. In *Proceedings of the 6th Annual Symposium on Discrete Algorithms (SODA)*, pages 604–613, 1995.

[19] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for the inversion distance between two permutations. In *Proc. of 4th Ann. Symp. on Combinatorial Pattern Matching*, Lecture Notes in Computer Science 684, pages 87–105. Springer Verlag, 1993.

[20] J. Kececioglu and D. Sankoff. Efficient bounds for oriented chromosome inversion distance. In *Proc. of 5th Ann. Symp. on Combinatorial Pattern Matching*, pages 307–325. Springer-Verlag LNCS 807, 1994.

[21] E. B. Knox, S. R. Downie, and J. D. Palmer. Chloroplast genome rearrangements and evolution of giant lobelias from herbaceous ancestors. *Mol. Biol. Evol.*, 10:414–430, 1993.

[22] D. Knuth. *The Art of Computer Programming, Vol. III: Sorting and Searching*. Addison-Wesley, Reading, MA, 1973.

[23] J. H. Nadeau and B. A. Taylor. Lengths of chromosomal segments conserved since divergence of man and mouse. *Proc. Natl. Acad. Sci. USA*, 81:814–818, 1984.

[24] R. Pinter and S. Skiena. Sorting with length-weighted reversals. In *Proc. 13th International Conference on Genome Informatics (GIW 2002)*, pages 103–111. Universal Academic Press, 2002.

[25] E. P. C. Rocha. An appraisal of the potential for illegitimate recombination in bacterial genomes and its consequences: From duplications to genome reduction. *Genome Res.*, 13(6a):1123–1132, 2003.

[26] A. Siepel. An algorithm to find all sorting reversals. In *Proc. Sixth Annual International Conference on Computational Molecular Biology (RECOMB'02)*, pages 281–290. ACM Press, 2002.

[27] A. H. Sturtevant and T.Dobzhansky. Inversions in the third chromosome of wild races of *drosophila pseudoobscura*, and their use in the study of the history of the species. *Proc. Nat. Acad. Sci.*, 22:448–450, 1936.

[28] F. Swidan, M. A. Bender, D. Ge, S. He, H. Hu, and R. Pinter. Sorting by length-weighted reversals: Dealing with signs and circularity. In S. Sahinalp, S. Muthukrishnan, and U. Dogrusoz, editors, *Proc. of the Fifteenth Annual Combinatorial Pattern Matching Symposium (CPM'04)*, volume 3109 of *Lecture Notes in Computer Science*, pages 32–46, Berlin, 2004. Springer Verlag.