Postkarte

CANADA

Data Structuring Group
Department of Computer Science
University of Waterloo
Prof. Derick Wood

Waterloo
Ontario N2L 3G1
Dear Sir,

I would greatly appreciate a reprint of your paper
- A New Measure of Resortedness
- Roughly Sorting: A Generalization of Sorting

from (CS 87-58 and CS 87-55)

and related papers, if you have copies for distribution.
Thanking you in advance, Yours sincerely

[Signature]

AUG 31 1988
<table>
<thead>
<tr>
<th><strong>DESCRIPTION</strong></th>
<th><strong>ACCOUNT NO.</strong></th>
<th><strong>NEGATIVES</strong></th>
<th><strong>QUANTITY</strong></th>
<th><strong>OPER. NO.</strong></th>
<th><strong>TIME</strong></th>
<th><strong>LABOUR CODE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>New Measure of Presortedness CS-87-58</td>
<td>1 2 6 6 1 7 6 8 1</td>
<td>FILM 1 0 0 0</td>
<td>1 0</td>
<td>FILM 1 0 0 0</td>
<td>1 0</td>
<td>FILM 1 0 0 0</td>
</tr>
<tr>
<td><strong>Packing</strong></td>
<td><strong>DESCRIPTION</strong></td>
<td><strong>NEGATIVES</strong></td>
<td><strong>QUANTITY</strong></td>
<td><strong>OPER. NO.</strong></td>
<td><strong>TIME</strong></td>
<td><strong>LABOUR CODE</strong></td>
</tr>
<tr>
<td><strong>ORDER</strong></td>
<td><strong>DESCRIPTION</strong></td>
<td><strong>NEGATIVES</strong></td>
<td><strong>QUANTITY</strong></td>
<td><strong>OPER. NO.</strong></td>
<td><strong>TIME</strong></td>
<td><strong>LABOUR CODE</strong></td>
</tr>
<tr>
<td><strong>DELIVER</strong></td>
<td><strong>DESCRIPTION</strong></td>
<td><strong>NEGATIVES</strong></td>
<td><strong>QUANTITY</strong></td>
<td><strong>OPER. NO.</strong></td>
<td><strong>TIME</strong></td>
<td><strong>LABOUR CODE</strong></td>
</tr>
<tr>
<td><strong>PICK-UP</strong></td>
<td><strong>DESCRIPTION</strong></td>
<td><strong>NEGATIVES</strong></td>
<td><strong>QUANTITY</strong></td>
<td><strong>OPER. NO.</strong></td>
<td><strong>TIME</strong></td>
<td><strong>LABOUR CODE</strong></td>
</tr>
<tr>
<td><strong>ITY CENTRE</strong></td>
<td><strong>DESCRIPTION</strong></td>
<td><strong>NEGATIVES</strong></td>
<td><strong>QUANTITY</strong></td>
<td><strong>OPER. NO.</strong></td>
<td><strong>TIME</strong></td>
<td><strong>LABOUR CODE</strong></td>
</tr>
<tr>
<td><strong>BINDERY</strong></td>
<td><strong>DESCRIPTION</strong></td>
<td><strong>NEGATIVES</strong></td>
<td><strong>QUANTITY</strong></td>
<td><strong>OPER. NO.</strong></td>
<td><strong>TIME</strong></td>
<td><strong>LABOUR CODE</strong></td>
</tr>
<tr>
<td><strong>OUTSIDE SERVICES</strong></td>
<td><strong>DESCRIPTION</strong></td>
<td><strong>NEGATIVES</strong></td>
<td><strong>QUANTITY</strong></td>
<td><strong>OPER. NO.</strong></td>
<td><strong>TIME</strong></td>
<td><strong>LABOUR CODE</strong></td>
</tr>
</tbody>
</table>

**Note:** This form is for printing requisition and graphic services, detailing specifications, quantities, and labor codes for various printing tasks. The form includes sections for packs, film, numbers, packing, order, delivery, pick-up, center, bindery, and outside services. Each task is listed with specific details and quantities.
A New Measure of Presortedness

Vladimir Estivill-Castro
Derick Wood

Data Structuring Group
Research Report CS-87-58

October 15, 1987
A New Measure of Presortedness *

Vladimir Estivill-Castro †  Derick Wood †

October 19, 1987

Abstract

A new measure of presortedness is presented which we call \( \text{Par}(X) \). It is proved that this measure is distinct from other common measures of presortedness. We design a comparison-based sorting algorithm that sorts arbitrary lists in \( O(n \log n) \) comparisons and, moreover, is optimal with respect to this new measure.

1 Introduction

We present a new measure of presortedness that is derived from the study of parallel-sorting algorithms [6]. This measure which is based on the idea of presortedness presented in [3] we call \( \text{Par}(X) \).

Intuitively, a list is \( p \)-sorted if, for each element \( x \) in the list, deleting the \( p \) elements immediately before and after \( x \), the resulting list has \( x \) in its correct sorted position. We first introduce the formal definitions and the fundamental properties of \( \text{Par}(X) \) in Section 2. Then, in Section 3, we compare \( \text{Par}(X) \) with the measures: \( \text{Inv}(X) \), the number of inversions in \( X \); \( \text{Runs}(X) \), the number of runs in \( X \); \( \text{Exc}(X) = |X| \)– the number of cycles in the permutation corresponding to \( X \), and \( \text{Rem}(X) \) the minimum number of elements that need to be removed to obtain a sorted list. It is proved that \( \text{Par}(X) \) is not equivalent to any of these measures. We say these functions measure presortedness since nearly sorted lists have small measure. Mehlhorn [5], who coined the term presorted, called a list presorted if it has a small \( \text{Inv}(X) \) value.

If \( m \) is a measure of presortedness, an \( m \)-optimal comparison-based algorithm is an algorithm that sorts all lists, but performs particularly well for lists having small \( m \). A definition for optimality with respect to \( \text{Inv}(X) \)

\*This work was supported under a Natural Sciences and Engineering Research Council of Canada Grant No. A-5692

†Data Structuring Group, Department of Computer Science, University of Waterloo, WATERLOO, Ontario N2L 3G1, CANADA
was suggested by Mehlhorn in [5], the idea was studied empirically by Cook and Kim [2], but the formal concept is due to Mannila [4]. An $m$-optimal algorithm performs optimally with respect to a measure of presortedness if it performs as well as any sorting algorithm that uses as input not only the list $X$ but also the value $m(X)$.

We present a Par-optimal algorithm in Section 5, while Section 4 provides a lower bound for any algorithm that sorts $p$-sorted lists of length $n$. Our algorithm is an $O(n \log n)$ comparison-based sorting algorithm that performs optimally with respect to the Par($X$) measure.

For a list $X$, $|X|$ denotes its length and for a set $S$, $|S|$ denotes its cardinality. Let $X = \langle x_1, \ldots, x_n \rangle$ and $Y = \langle y_1, \ldots, y_m \rangle$ be two lists; then their catenation is denoted by $XY$ and is defined as $\langle x_1, \ldots, x_n, y_1, \ldots, y_m \rangle$. We denote the empty list as $\langle \rangle$.

2 Definitions

Let $X = \langle x_1, x_2, \ldots, x_n \rangle$ be a list of length $n$ of elements $x_i$ from some linear order, that is, for all $i, j \in \{1, 2, \ldots, n\}$, $x_i \leq x_j$ or $x_j \leq x_i$. We also call $X$ an $n$-list.

**Definition 2.1** $X$ is $p$-sorted if and only if, for all $i, j \in \{1, 2, \ldots, n\}$, $i - j > p$ implies $x_j \leq x_i$.

The following are immediate properties of $p$-sortedness:

1. If a list is $p$-sorted, then it is $(p + i)$-sorted, for all $i \geq 0$.
2. $p$-sorted does not imply $(p - 1)$-sorted.
3. A list is 0-sorted if and only if it is sorted (completely sorted).
4. The definition of being $p$-sorted is equivalent to:
   $X$ is $p$-sorted if and only if the following two conditions are satisfied:
   
   (a) There is no $j < i - p$ such that $x_i < x_j$.
   (b) There is no $j > i + p$ such that $x_i > x_j$.

**Definition 2.2** Let $X = \langle x_1, \ldots, x_n \rangle$ be a list. $Y$ is said to be an $m$-subsequence of $X$ if $Y = \langle x_{i(1)}, x_{i(2)}, \ldots, x_{i(m)} \rangle$ and $i : \{1, 2, \ldots, m\} \rightarrow \{1, 2, \ldots, n\}$ is injective and monotonically increasing. We say that $Y$ is an $m$-sublist of a $n$-list $X$ if $Y = \langle x_i, x_{i+1}, \ldots, x_{i+m-1} \rangle$, for some $i$, $1 \leq i \leq n - m + 1$.

Igarashi and Wood [3] provided a useful equivalent local condition for $p$-sortedness.
Theorem 2.1  A list \(X\) is \(p\)-sorted, if and only if, every \((2p + 2)\)-sublist of \(X\) is \(p\)-sorted. Moreover, for every \(p \geq 0\), there is a list \(X\) satisfying the following two conditions:

1. \(X\) is not \(p\)-sorted.
2. Every \((2p + 1)\)-sublist of \(X\) is \(p\)-sorted.

This theorem is the fundamental tool to prove that a given list is \(p\)-sorted.

In the examples and the following definitions we assume without loss of generality that the \(x_i\) are nonnegative integers.

We now define the new measure of presortedness.

Definition 2.3  Let \(N^{<N}\) denote the set of all finite sequences of nonnegative integers. Define \(Par: N^{<N} \rightarrow N\) by:

\[
Par(X) = p \text{ if and only if } p = \min\{q | X \text{ is } q\text{-sorted}\}.
\]

The following are some basic properties of \(Par(X)\):

1. For all \(X = (x_1, x_2, \ldots, x_n)\), \(0 \leq Par(X) \leq n - 1\).
2. \(Par(X) = 0\) if and only if \(X\) is sorted.
3. For all \(X = (x_1, x_2, \ldots, x_n)\), \(Par(X) = n - 1\) if and only if \(x_n < x_1\).

We now give the five axioms, proposed by Mannila [4], that a measure of presortedness must satisfy and we verify that \(Par\) satisfies them.

Definition 2.4  Letting \(m: N^{<N} \rightarrow N\) be some function, we say that \(m\) is a measure of presortedness, if and only if:

1. If \(X\) is in ascending order, \(m(X) = 0\).
2. If \(X = (x_1, x_2, \ldots, x_n)\), \(Y = (y_1, y_2, \ldots, y_n)\) and \(x_i \leq x_j\) if and only if \(y_i \leq y_j\) for all \(i, j \in \{1, 2, \ldots, n\}\), then \(m(X) = m(Y)\).
3. If \(Y\) is a subsequence of \(X\) then \(m(Y) \leq m(X)\).
4. If \(X \leq Y\) (that is, every element of \(X\) is no greater than every element of \(Y\)), then \(m(XY) \leq m(X) + m(Y)\).
5. For all \(x\) in \(N\), \(m((x)X) \leq |X| + m(X)\).

Lemma 2.2  \(Par(X)\) is a measure of presortedness.
Proof: We verify the five axioms for \( \text{Par}(X) \).

1. This was already established as an immediate property.

2. Suppose \( \text{Par}(X) = p \) and \( \text{Par}(Y) = q \) and, without loss of generality, assume \( p < q \). Since \( Y \) is not \((q-1)\)-sorted, \((q)\) must be positive, since \( 0 \leq p < q \) there exist \( y_i \) and \( y_j \) such that \( i - j = q \), and \( y_i > y_j \). This implies \( z_i > z_j \) and \( i - j = q > p \). Since \( X \) is \( p \)-sorted we must have \( z_i \leq z_j \), and we have obtained a contradiction.

3. Let \( X = \langle x_1, \ldots, x_n \rangle \) and \( Y = \langle x_{i(1)}, \ldots, x_{i(s)} \rangle \), for some \( s \), \( 1 \leq s \leq n \), where \( i\{1, \ldots, s\} \rightarrow \{1, \ldots, n\} \) is such that \( 1 \leq j < k \leq s \) implies \( i(j) < i(k) \). We wish to prove that \( k - j > p \) implies \( z_{i(j)} \leq z_{i(k)} \). Since \( i \) is a bijection and monotonically increasing, \( k - j \leq i(j) - i(k) \). Hence, \( p < k - j \) implies \( p < i(k) - i(j) \). In other words, \( z_{i(j)} \leq z_{i(k)} \) as desired.

4. If \( X < Y \) we claim that \( \text{Par}(XY) = \max\{\text{Par}(X), \text{Par}(Y)\} \). Let \( \text{Par}(X) = p, \text{Par}(Y) = q \) and \( r = \max\{p, q\} \). We first prove that \( \text{Par}(XY) \leq r \). We write \( XY = \langle z_1, z_2, \ldots, z_{n+m} \rangle \), where \( z_i = z_k \) if \( 1 \leq i \leq n \) and \( z_i = y_{i-n} \) if \( n+1 \leq i \leq n+m \). We must show that if \( i - j > r \), then \( z_j \leq z_i \). Suppose \( i - j > r \).

Case 1: If \( i \leq n \) then \( j \leq i \) and \( z_j = z_j \leq z_i = z_i \), because \( \text{Par}(X) = p \leq r \).

Case 2: If \( n+1 \leq j \), then \( z_i = y_{i-n} \), \( z_j = y_{j-n} \), and \( (i-n) - (j-n) = i - j \).

Thus \( z_j \leq z_i \), because \( \text{Par}(Y) = q \leq r \).

Case 3: If \( j \leq n \) and \( n+1 \leq i \), then \( z_j \in X \) and \( z_i \in Y \). Since \( X \leq Y \), we conclude that \( z_j \leq z_i \).

Notice that \( \text{Par}(XY) \leq \max\{\text{Par}(X), \text{Par}(Y)\} \) is enough to verify the axiom since \( \text{Par}(X) \geq 0 \) and \( \text{Par}(Y) \geq 0 \). The reader can now verify that \( \text{Par}(XY) \geq \max\{\text{Par}(X), \text{Par}(Y)\} \).

5. By the first basic property, \( \text{Par}(\langle x \rangle X) \leq |\langle x \rangle X| - 1 = |X| \leq \text{Par}(X) + |X| \), since \( \text{Par}(X) \geq 0 \). \( \square \)

The concept of an optimal algorithm with respect to a measure of presortedness was given in a general form by Mannila [4].

**Definition 2.5** Let \( m \) be a measure of presortedness, and \( S \) a sorting algorithm which uses \( T_S(X) \) comparisons on input \( X \). We say that \( S \) is optimal with respect to \( m \) (or \( m \)-optimal) if, for some \( c > 0 \), we have, for all \( X = \langle x_1, x_2, \ldots, x_n \rangle \):

\[
T_S(X) \leq c \cdot \max\{|X|, \log(||\text{below}(X, m)||)\}
\]

where \( \text{below}(X, m) = \{\pi | \pi \text{ is a permutation of } \{1, \ldots, n\} \text{ and } m(\pi(X)) \leq m(X)\} \).
3 Comparing Measures of Presortedness

We now compare our measure of presortedness with other measures.

Definition 3.1 The number of inversions of $X = (x_1, \ldots, x_n)$ is denoted by $\text{Inv}(X)$ and defined by:

$$
\text{Inv}(X) = \|\{(i, j) | 1 \leq i < j \leq n \text{ and } x_i > x_j\}\|.
$$

Lemma 3.1 For all $X \in N^{<N}$, $\text{Par}(X) \leq \text{Inv}(X)$.

Proof: Let $\text{Par}(X) = p$. If $p = 0$, $X$ is sorted, so $\text{Inv}(X) = 0$ and we are done.

If $p \neq 0$, since $X$ is not $(p - 1)$-sorted, there exists $x_i$ such that $x_{i+p} < x_i$. Now consider the $p - 1$ elements $x_{i+1}, x_{i+2}, \ldots, x_{i+p-1}$. If $x_{i+s}$ with $s \in \{1, 2, \ldots, p - 1\}$ is such that:

Case 1: $x_{i+s} > x_i$. Then $x_{i+s} > x_i > x_{i+p}$ and $i + p > i + s$, so we have an inversion.

Case 2: $x_{i+s} < x_i$. Then $i + s > i$ and we have an inversion.

For each $s \in \{1, 2, \ldots, p - 1\}$ we have an inversion and since $x_{i+p} < x_i$, we have at least $p$ inversions. Therefore $\text{Inv}(X) \geq p = \text{Par}(X)$. \qed

Lemma 3.2 There is no $c > 0$ such that, for all $X \in N^{<N}$, $\text{Inv}(X) \leq c \cdot \text{Par}(X)$.

Proof: Let $X = (n, n-1, \ldots, 1)$. Then $\text{Inv}(X) = n(n-1)/2$ which is $\Theta(n^2)$ while $\text{Par}(X) = n - 1$. \qed

Definition 3.2 The number of maximal ascending subsequences of $X$ is $\|\{i | 1 \leq i < n \text{ and } x_{i+1} < x_i\}\| + 1$. Since this is trivially not a measure of presortedness, we define $\text{Runs}(X)$ to be this value less one.

Lemma 3.3 1. There is no $c > 0$ such that, for all $X \in N^{<N}$, $\text{Par}(X) \leq c \cdot \text{Runs}(X)$.

2. There is no $c > 0$ such that, for all $X \in N^{<N}$, $\text{Runs}(X) \leq c \cdot \text{Par}(X)$.

Proof: 1. Let $X = (n, 2, 3, \ldots, n-1, 1)$ then $\text{Par}(X) = n - 1$ but $\text{Runs}(X) = 2$.

2. Consider $X = (2, 1, 4, 3, 6, 5, \ldots, n, n - 1)$ then $\text{Par}(X) = 1$ while $\text{Runs}(X) = \lfloor n/2 \rfloor$. \qed
Definition 3.3 The length of the largest ascending subsequence is denoted by \( \text{Las}(X) \) and is defined by: \( \text{Las}(X) = \max \{ t | \exists i(1), i(2), \ldots, i(t) \mid 1 \leq i(1) < i(2) < \cdots < i(t) \leq n \text{ and } x_{i(1)} < \cdots < x_{i(t)} \} \). Since \( \text{Las}(X) \neq 0 \) when \( X \) is sorted, we define \( \text{Rem}(X) = |X| - \text{Las}(X) \). \( \text{Rem}(X) \) is a measure of presortedness.

Lemma 3.4 For all lists \( X \) of length \( n \), \( \text{Las}(X) \geq \lceil n/(\text{Par}(X) + 1) \rceil \).

Proof: Consider \( x_1, x_{1+p+1}, x_{1+2(p+1)}, \ldots, x_{1+k(p+1)} \) with \( 1 + k(p+1) \leq n < 1 + (k+1)(p+1) \). If \( X \) is \( p \)-sorted, the above sequence is in ascending order and has length \( k+1 \), but

\[
n < 1 + (k+1)(p+1)
\]

\[
\Rightarrow n \leq (k+1)(p+1)
\]

\[
\Rightarrow n/(p+1) \leq k+1 \leq \text{Las}(X)
\]

and \( \text{Las}(X) \) is an integer so \( \text{Las}(X) \geq \lceil n/(p+1) \rceil \).

Now consider the list \( X = \langle p+1, p, p-1, \ldots, 2, 1, 2(p+1), 2(p+1)-1, \ldots \rangle \). This list is easily seen to be \( p \)-sorted and not \( (p-1) \)-sorted (using Theorem 2.1), so \( \text{Par}(X) = p \) and \( \text{Las}(X) = \lceil n/(p+1) \rceil \), proving that the bound of the above Lemma is tight. On the other hand \( X = \langle p+1, 1, 2, 3, \ldots, p, 2(p+1), 2(p+1)-p, 2(p+1)-p+1, 2(p+1)-p+2, \ldots, 2(p+1)-1, 3(p+1), 3(p+1)-p, 3(p+1)-p+1, \ldots \rangle \) is such that \( \text{Par}(X) = p \) and \( \text{Las}(X) \geq p\lfloor n/(p+1) \rfloor \). This shows that, in general, we can have examples with strict inequality and with \( \text{Las}(X) \) being far from \( \lceil n/(\text{Par}(X) + 1) \rceil \). □

Lemma 3.5 1. \( \text{Rem}(X) \leq |X|(1 - 1/(\text{Par}(X) + 1)) \).

2. There is no \( c > 0 \) such that, for all \( X \in N < N \), \( \text{Par}(X) \leq c \cdot \text{Rem}(X) \).

3. There is no \( c > 0 \) such that, for all \( X \in N < N \), \( \text{Rem}(X) \leq c \cdot \text{Par}(X) \).

Proof: 1. \( \text{Rem}(X) = |X| - \text{Las}(X) \leq |X| - |X|/(\text{Par}(X) + 1) \).

2. Let \( X = \langle n, 2, 3, \ldots, n-1, 1 \rangle \) then \( \text{Rem}(X) = 2 \) while \( \text{Par}(X) = n-1 \).

3. Let \( X = \langle 2, 1, 4, 3, \ldots \rangle \) then \( \text{Rem}(X) = \lfloor n/2 \rfloor \) but \( \text{Par}(X) = 1 \). □

Definition 3.4 We now consider \( \text{Exc}(X) = n - \text{the number of cycles in the permutation of } \{1, 2, \ldots, n\} \text{ corresponding to } X \). \( \text{Exc} \) is also a measure of presortedness.

Lemma 3.6 There is no \( c > 0 \) such that, for all \( X \in N < N \), \( \text{Par}(X) \leq c \cdot \text{Exc}(X) \) and there is no \( d > 0 \) such that, for all \( X \in N < N \), \( \text{Exc}(X) \leq d \cdot \text{Par}(X) \).
Proof: Let \( X = (n, 2, 3, \ldots, n-1, 1) \) then the cycles of the permutation are \( (1 \ n)(2)(3) \ldots(n-1) \) and then \( \text{Exc}(X) = 1 \) while \( \text{Par}(X) = n - 1 \), on the other hand if \( X = (2, 1, 4, 3, 6, 5, \ldots, n, n-1) \) we have \( \text{Par}(X) = 1 \) but \( \text{Exc}(X) \geq \lceil n/2 \rceil \).


We have compared the measure \( \text{Par}(X) \) with the most common measures of presortedness and have shown:

**Theorem 3.7** \( \text{Par}(X) \) is not equivalent to any of the measures of presortedness \( \text{Inv}(X) \), \( \text{Rem}(X) \), \( \text{Runs}(X) \) or \( \text{Exc}(X) \), but, for all \( X \in \mathcal{N}^N \):

1. \( \text{Par}(X) \leq \text{Inv}(X) \).
2. \( \text{Rem}(X) \leq |X|(1 - 1/(\text{Par}(X) + 1)) \).

We conclude that \( \text{Par}(X) \) measures global presortedness and does not recognize local presortedness. In this sense \( \text{Par}(X) \) is similar to \( \text{Inv}(X) \).

4 A Lower Bound

We claim that any comparison-based algorithm that sorts any \( p \)-sorted list of length \( n \) requires \( \Omega(\max\{n, n \log(p+1)\}) \) comparisons.

To show this, consider the set

\[
A_i = \{1+i(p+1), 2+i(p+1), 3+i(p+1), \ldots, p+i(p+1), p+1+i(p+1)\}
\]

Let \( B_i \) be any permutation of the elements of \( A_i \). We build a list \( X \) by catenation of the \( B_i \):

\[
X = B_0B_1B_2\cdots B_{\lceil n/(p+1) \rceil - 1}
\]

It can be directly verified that \( X \) is \( p \)-sorted. Further, this shows that there are at least \( (p+1)!^{\lceil n/(p+1) \rceil} \) \( p \)-sorted lists. Hence, we conclude:

**Theorem 4.1** There are at least \( (p+1)!^{\lceil n/(p+1) \rceil} \) \( n \)-lists \( X \) with \( \text{Par}(X) \leq p \).

Therefore, any comparison-based algorithm that sorts \( p \)-sorted lists requires at least \( \lfloor n/(p+1) \rfloor \log((p+1)!)) \) comparisons, that is, \( \Omega(n \log(p+1)) \) comparisons.

5 An Optimal Algorithm

We propose a comparison-based algorithm called the Try-to-Merge Sort that given a list as input produces the corresponding sorted list as output.
Try-to-Merge Sort is an $O(n \log n)$ worst-case sorting algorithm, but if the input list $X$ is $p$-sorted, then we can certify that the algorithm requires $O((\log(p+1)+1)n)$ comparisons.

Letting $X = \langle x_1, \ldots, x_n \rangle$ define: $X_{\text{even}} = \langle x_2, x_4, \ldots, x_{2\lfloor n/2 \rfloor} \rangle$ and $X_{\text{odd}} = \langle x_1, x_3, \ldots, x_{2\lceil n/2 \rceil} \rangle$. We claim:

**Theorem 5.1**

1. If $X$ is $p$-sorted then $X_{\text{even}}$ and $X_{\text{odd}}$ are $\lfloor p/2 \rfloor$-sorted. Moreover, for every $p > 1$, there is a $p$-sorted list $X$ such that $X_{\text{even}}$ and $X_{\text{odd}}$ are not $(\lfloor p/2 \rfloor - 1)$-sorted.

2. For any $n \in N$, there is a list $X$ such that $\text{Par}(X) > n$ and $X_{\text{even}}$, $X_{\text{odd}}$ are both sorted.

**Proof:** 1. Let $b_i$ and $b_j$ be elements of $X_{\text{even}}$ such that $j - i > \lfloor p/2 \rfloor$. We must prove that $b_j \geq b_i$. Now $b_j = x_{2j}, b_i = x_{2i}$, and $2j - 2i = 2(j - i)$. Since $j - i$ is an integer, $j - i > p/2$, and we obtain $2j - 2i > 2(p/2) = p$. Thus, since $X$ is $p$-sorted, $x_{2j} \geq x_{2i}$, that is, $b_j \geq b_i$. The claim for $X_{\text{odd}}$ is proved similarly.

Now, let $p > 1$. If $p$ is odd, let $X = \langle x_1, x_2, \ldots, x_{p+3} \rangle$, where $x_i = 2 + i$, for $i = 1, 2, \ldots, p - 1, x_p = 1, x_{p+1} = 2, x_{p+2} = 3 + p + 2$, and $x_{p+3} = 3 + p + 3$, and if $p$ is even, let $X = \langle x_1, \ldots, x_{p+4} \rangle$, where $x_i = 2 + i$, for $i = 1 \ldots p, x_{p+1} = 1, x_{p+2} = 2, x_{p+3} = 3 + p + 3$, and $x_{p+4} = 3 + p + 4$. We claim that $X$ is $p$-sorted and $X_{\text{even}}$ and $X_{\text{odd}}$ are not $\lfloor (p/2) - 1 \rfloor$-sorted.

Suppose $p$ is even, $(p = 2k)$, $x_1$ and $x_{p+1}$ belong to $X_{\text{odd}}$, more precisely, $x_1$ is the first element of $X_{\text{odd}}$ and $x_{p+1}$ is the $\lceil (p+1)/2 \rceil$ element in $X_{\text{odd}}$. But $\lceil (p+1)/2 \rceil - 1 = k > \lfloor p/2 \rfloor - 1$, and $x_1 = 2 + 1 = 3 > 1 = x_{p+1}$, hence $X_{\text{odd}}$ is not $(\lfloor p/2 \rfloor - 1)$-sorted. All other cases are verified similarly.

2. Finally, let $n \in N$ and define $X = \langle x_1, \ldots, x_{2n+3} \rangle$ by: $x_{2i} = i$, for $i = 1, 2, \ldots, n + 1$, and $x_{2i-1} = n + 2i$, for $i = 1, 2, \ldots, n + 2$. It can be verified that $\text{Par}(X) = 2n + 1$ and $X_{\text{even}}$ and $X_{\text{odd}}$ are sorted. 

We assume that we have a boolean function $\text{merge}(X_1, X_2, X)$ that attempts to merge the two lists $X_1$ and $X_2$ as if they are sorted. If $X_1$ and $X_2$ are sorted, it returns $\text{true}$ and their merge is $X$, otherwise it returns $\text{false}$ and $X$ is undefined. If the input lists $X_1$ and $X_2$ have lengths $n_1$ and $n_2$, then the merging algorithm has complexity $O(n_1 + n_2)$.

### 5.1 Sorting Algorithm

**Try-to-Merge Sort**($X$ : list, $n$ : integer) \{ $|X| = n$ \}

**Input:** $X = \langle x_1, x_2, \ldots, x_n \rangle$.

**Output:** The elements in $X$ in ascending order.

**begin**
if merge($X_{even}, X_{odd}, X$) then \{successful\}
else
begin
Try-to-Merge Sort($X_{even}, \lceil n/2 \rceil$);
Try-to-Merge Sort($X_{odd}, \lfloor n/2 \rfloor$);
merge($X_{even}, X_{odd}, X$)
end

end

5.2 Algorithm Correctness

We assume that the procedure that performs the merge operation is correct.

Lemma 5.2 For all lists $X$ of length $n$, Try-to-Merge Sort($X, n$) returns $X$ in sorted order.

Proof: We prove correctness by induction on the length of $X$.

Basis: If $|X| = 0$ then $X = \langle \rangle$, so $X_{even}$ and $X_{odd}$ are also empty, and as $merge(X_{even}, X_{odd}, X)$ is true, it yields $X = \langle \rangle$. Clearly the empty list is the correct result.

If $|X| = 1$ then $X_{odd} = X$ and $X_{even} = \langle \rangle$, $merge(X_{even}, X_{odd}, X)$ is true. It returns $X_{odd}$ which is the correct result since a list of length one is always sorted.

Induction Step: Assume that the algorithm works correctly for input lists of length less than $n$, and we are given $X$ of length $n \geq 2$. By the assumptions about procedure $merge(X_1, X_2, X)$ we have two cases according to whether this procedure returns true or false.

Case 1: If the merge is successful, that is, procedure $merge$ returns true, then it returns $X$ is sorted order.

Case 2: If the merge is unsuccessful, the algorithm performs the 'else' part. Since $n \geq 2$, $n > \lceil n/2 \rceil \geq \lfloor n/2 \rfloor$ and, by the induction hypothesis, the recursive calls to Try-to-Merge Sort return $X_{even}$ and $X_{odd}$ in sorted order. Hence, in the inner call to procedure $merge$ they will be merged successfully producing as output the lists of elements in $X$ in sorted order.

\[\square\]
5.3 Algorithm Complexity

We now show that if we execute Try-to-Merge Sort$(X, n)$, where $n$ is the length of the list $X$, then the algorithm performs $O(n \log n)$ comparisons in the worst case.

By a worst case analysis the number of comparisons that the algorithm performs satisfies the recurrence relation:

\[
T(1) = 1 \\
T(n) = 2T(n/2) + cn, \text{ for some constant } c > 0.
\]

This has growth rate $\Theta(n \log n)$; see [1].

5.4 Par$(X)$-optimality

We claim that

**Lemma 5.3** If $X$ is $(2^{i} - 1)$-sorted, then the maximum depth of recursion of Try-to-Merge Sort$(X, |X|)$ is $i$.

**Proof:** We prove this claim by induction on $i$.

**Basis:** If $i = 0$ then $X$ is $2^0 - 1 = 0$-sorted. In this case $X$ is sorted, so $X_{\text{even}}$ and $X_{\text{odd}}$ are sorted, therefore the merging is successful. $X$ is returned in ascending order and no recursive calls are made.

**Induction Step:** Assume that, for some $i \geq 1$, if $k < i$ and $X$ is $(2^k - 1)$-sorted, then the depth of recursion of the call Try-to-Merge Sort$(X, |X|)$ is no greater than $k$.

Let $X$ be $(2^{i} - 1)$-sorted, and suppose we call Try-to-Merge Sort$(X, |X|)$. If the merging on the odd and even subsequences of $X$ is successful, then we produce the desired result with no recursive calls. But, in the worst case, the merging is unsuccessful and we call recursively:

1. Try-to-Merge Sort$(X_{\text{even}}, |X|/2)$
2. Try-to-Merge Sort$(X_{\text{odd}}, |X|/2)$

By Theorem 5.1 $X$ being $(2^{i} - 1)$-sorted implies that $X_{\text{even}}$ is $\left\lfloor \frac{2^{i} - 1}{2} \right\rfloor$-sorted, that is, $X_{\text{even}}$ is $\left[2^{i-1} - 1/2\right] = (2^{i-1} - 1)$-sorted, and similarly $X_{\text{odd}}$ is $(2^{i-1} - 1)$-sorted. By the induction hypothesis the above two recursive calls have a depth of recursion of at most $i - 1$. Therefore the maximum depth of recursion of the original call is bounded by $i$.

This completes our proof. \[\square\]

Using this lemma, and the observation that at each level of recursion we perform at most $|X|$ comparisons, we conclude:

**Theorem 5.4** If $X$ is $p$-sorted and $|X| = n$, then Try-to-Merge Sort$(X, n)$ requires $O\left(\left\lfloor \log(p + 1) + 1\right\rfloor n\right)$ comparisons in the worst case.
To prove that *Try-to-Merge Sort* is *Par*-optimal we need:

**Lemma 5.5** There is a $d > 0$ such that, for all $n > 1$ and for all $m$ with $1 < m \leq n$,

$$\log(m) + 1 \leq d \lfloor n/m \rfloor \frac{\log(m!)}{n}$$

**Proof:** We claim that $d = \max\{8, 4\log(3)+1\}$

Let $k = \lfloor n/m \rfloor$. Then $n/m < k + 1$, so $1/n > 1/m(k+1)$, $n/m \geq 1$, and $k/(k+1) \geq 1/2$. Hence,

$$d \lfloor n/m \rfloor \frac{\log(m!)}{n} > d \frac{k}{k+1} \frac{\log(m!)}{m} \geq d \frac{\log(m!)}{m}$$

**Case 1:** $m = 2$.

Since $d \geq 8$, $\frac{d \log(m!)}{m} \geq 2 = \log(m) + 1$ as claimed.

**Case 2:** $m \geq 3$.

$$\frac{d \log(m!)}{2 \cdot m} = \frac{d}{2} \log((m!)^{1/m}) > \frac{d}{4} (\log(m) - \log 2) = (d/4)(\log(m) - 1)$$

By the definition of $d$ and because $m \geq 3$

$$(d/4)(\log(m) - 1) \geq \frac{\log(3)+1}{\log(3)-1} (\log(m) - 1) \geq \log(m) + 1$$

$\square$

**Theorem 5.6** *Try-to-Merge Sort* is *Par*-optimal.

**Proof:** Let $T_{DMS}(X)$ be the number of comparisons that *Try-to-Merge Sort* performs on input $X$. Since, by Lemma 4.1, there are at least $(\text{Par}(X) + 1)!(||X||/(\text{Par}(X) + 1))$ lists in $\text{below}(X, \text{Par})$, we have

$$||X||/(\text{Par}(X) + 1)! \log((\text{Par}(X) + 1)!) \leq \log(||\text{below}(X, \text{Par})||) \quad (1)$$

and by Theorem 5.4 there is an $e > 0$ such that, for all $X \in N^{<N}$,

$$T_{DMS}(X) \leq e(\log(\text{Par}(X) + 1) + 1)||X|| \quad (2)$$

Let $X \in N^{<N}$.
Case 1: \( \text{Par}(X) = 0 \). Since \( X \) is sorted if and only if \( \text{Par}(X) = 0 \)

\[
\log(||\text{below}(X, \text{Par})||) = \log(||\{X\}||) = \log(1) = 0
\]

Moreover, in this case, the algorithm performs a successful merge. Hence, there is a \( c_1 > 0 \) independent of \( X \) such that

\[
T_{DMS}(X) \leq c_1 |X| = c_1 \max\{|X|, \log(||\text{below}(X, \text{Par})||)\}
\]

Case 2: \( \text{Par}(X) > 0 \). By Lemma 5.5, there is a \( d > 0 \) such that, for all \( X \) where \( n = |X| \) and \( m = \text{Par}(X) + 1 \),

\[
\log(\text{Par}(X) + 1) + 1 \leq d\left[\frac{|X|}{\text{Par}(X) + 1}\right] \log((\text{Par}(X) + 1)!)
\]

Thus, there is a \( d > 0 \) such that

\[
e(d\log(\text{Par}(X)+1)+1)|X| \leq e\cdot d\left[\frac{|X|}{\text{Par}(X) + 1}\right] \log((\text{Par}(X)+1)!) \quad (3)
\]

and by (1), (2), and (3), we conclude that there is a \( c_2 = e\cdot d > 0 \) such that

\[
T_{DMS}(X) \leq c_2 \max\{|X|, \log(||\text{below}(X, \text{Par})||)\}
\]

Setting \( c = \max\{c_1, c_2\} \), we conclude that there is a \( c > 0 \) such that, for all \( X \in N^N \),

\[
T_{DMS}(X) \leq c \max\{|X|, \log(||\text{below}(X, \text{Par})||)\}
\]

and the theorem is proved. \( \square \)

References


