University of Colorado at Boulder

Department of Computer Science

ECOT 7-7 Engineering Center
Campus Box 430
Boulder, Colorado 80309-0430
(303) 492-3902

June 13, 1988

Attention Technical Reports
Department of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1
Canada

I would like to obtain copies of the following papers:

1. W. H. Cheung, J. P. Black and E. G. Manning. A Study of Distributed Debugging. Research report CS-87-53, University of Waterloo. Department of Computer Science., 1987.
2. Y. Igarashi and D. Wood. Roughly Sorting: A Generalization of Sorting. Research report CS-87-55, University of Waterloo. Department of Computer Science., 1987.

If you could send me copies of these papers, I would appreciate it.

Sincerely,

Michael F. Schwartz
Assistant Professor

Prof. A. Myhrhof
Dept. Wiskunde - Informatica
[redacted] Vrije Universiteit Brussel
Pleinlaan 2 F7
[illegible] Brussel
~~Belgium~~ BELGIUM

sent for Derick Feb. 9/88

87-55/58/56/68

Rawlins / Baeza-Y. / Cubr.

UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT

Roughly Sorting: A Generalization of Sorting

Yoshihide Igarashi
Derick Wood

Data Structuring Group
Research Report CS-87-55

September, 1987

Roughly Sorting: A Generalization of Sorting*

Yoshihide Igarashi † Derick Wood ‡

September 25, 1987

Abstract

We study roughly sorting and roughly sorted sequences. A sequence $\alpha = (a_1, \ldots, a_n)$ is *k-sorted* if and only if α is a sequence satisfying the conditions: (1) there is no j such that $j < i - k$ and $a_i < a_j$ and (2) there is no j such that $j > i + k$ and $a_i > a_j$. A 0-sorted sequence is a sorted sequence in the usual sense. We first give a local characterization of k-sorted sequences and then design a k-sorting algorithm which we call *k-bubble sort*, since it is a generalization of bubble sort. An algorithm for sorting k-sorted sequences is also designed. For each $k \geq 1$, a lower bound on the number of comparisons needed to sort k-sorted sequences is obtained by a decision tree argument. We show that $0.69428n$ and $1.1265n$ are average case lower bounds on the numbers of comparisons for sorting 1-sorted and 2-sorted sequences of length n, respectively.

1 Introduction

Sorting is an essential part of data processing and algorithm design. However, in some applications we are asked to nearly or roughly sort sequences rather than to completely sort them. There are also some applications in which efficient algorithms for completely sorting nearly sorted sequences are required. The concept of roughly sorting has appeared in the papers by Sado and Igarashi [7,8] on parallel sorting on a mesh-connected processor array. They have designed fast parallel sorting algorithms in which a method of iteratively merging roughly sorted subfiles is incorporated. The notions of

*Part of the work was carried out while the first author was a visitor in the Department of Computer Science, University of Waterloo. This visit was supported by Japan Society for the Promotion of Science and the Natural Sciences and Engineering Research Council of Canada. The work of the second author was supported under a Natural Sciences and Engineering Research Council of Canada Grant No. A-5692

†Department of Computer Science, Gunma University, KIRYU 376, JAPAN

‡Data Structuring Group, Department of Computer Science, University of Waterloo, WATERLOO, Ontario N2L 3G1, CANADA

presorted and nearly sorted lists discussed in [2,5,6] are related concepts, but different from roughly sorted lists, as [3] prove.

In this paper, we study roughly sorting and roughly sorted sequences. Roughly sorting is a generalization of sorting. In Section 2, we formalize our notion of roughly sortedness giving k-sorted sequences and we give a local characterization of k-sorted sequences. In Section 3, we design a k-sorting algorithm that is a generalization of bubble sort. In Section 4, we design an algorithm for sorting k-sorted sequences and an algorithm for merging two k-sorted sequences to give a sorted sequence. In Section 5, we show that the worst case optimal number of item comparisons for sorting 1-sorted sequences of length n is $n-1$. For each k we estimate the number of k-sorted sequences of length n. From these estimates and a decision tree argument we derive lower bounds on the number of comparisons needed to sort k-sorted sequences.

2 Roughly Sortedness

We begin by formalizing our notion of roughly sorted sequences.

Definition 2.1 *A sequence $\alpha = (a_1, \ldots, a_n)$ is* k-sorted *if and only if the following two conditions are satisfied:*

1. *For all j such that $1 \leq j < i - k$, $a_j \leq a_i$.*
2. *For all j such that $i + k < j \leq n$, $a_i \leq a_j$.*

From the above definition α is sorted if and only if α is 0-sorted. We now introduce the notion of a b-block which is important for the following results.

Definition 2.2 *Given a sequence $\alpha = (a_1, \ldots, a_n)$, a non-negative integer b, and an integer i, $1 \leq i \leq n - b + 1$, the* b-block of α at position i *is the subsequence $(a_i, \ldots, a_{i+b-1})$ of α. A* b-block of α *is a b-block at some position i.*

As is well known, 0-sorted sequences can be characterized by the following local condition:

> $\alpha = (a_1, \ldots, a_n)$ is sorted if and only if for every i, $1 \leq i \leq n-1$, $a_i \leq a_{i+1}$, that is, $\alpha = (a_1, \ldots, a_n)$ is 0-sorted if and only if every 2-block is 0-sorted.

This local characterization of sorted sequences can be generalized as follows.

Theorem 2.1 *Let $\alpha = (a_1, \ldots, a_n)$ and k be a nonnegative integer. Then α is k-sorted if and only if every $(2k+2)$-block of α is k-sorted.*

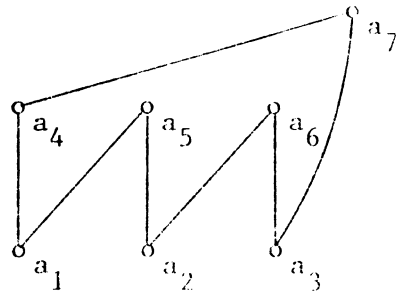

Figure 1: A Hasse diagram specified by the 2-sorted condition.

Proof: It is immediate that every $(2k+2)$-block of a k-sorted sequence is k-sorted. Therefore suppose that every $(2k+2)$-block of α is k-sorted. Let $(a_i, \ldots, a_{i+2k+1})$ be a $(2k+2)$-block of α. If $i+2k+1 < n$, $a_{i+k+1} \leq a_{i+2k+2}$ since $(a_{i+1}, \ldots, a_{i+2k+2})$ is k-sorted. Since $a_i \leq a_{i+k+1}$, we obtain $a_i \leq a_{i+2k+2}$. Clearly, this argument holds for any $2k+2$-block $(a_{i+j}, \ldots, a_{i+2k+j+1})$, where $1 \leq j < n-i-2k$. Therefore, for all t such that $i+k+1 \leq t \leq n$, we have $a_i \leq a_t$. A similar argument obtains for $1 \leq t \leq i-k-1$; hence, α satisfies the two conditions of Definition 1. □

One natural question is whether $(2k+2)$-blocks are necessary in the above theorem; that is, does it hold for $(2k+1)$-blocks, for example? This is not the case as the following theorem demonstrates.

Theorem 2.2 *For every k, $k \geq 0$, there exists a sequence α satisfying the following two conditions:*

1. *α is not k-sorted.*
2. *Every $(2k+1)$-block of α is k-sorted.*

Proof: Suppose that every $(2k+1)$-block of $\alpha = (a_1, \ldots, a_n)$ is k-sorted. This implies that the relative order of a_i and a_{i+2k+1} is not specified, for all i, $1 \leq i \leq n-2k-1$. Therefore, if $n \geq 2k+2$, we can assign an appropriate value to each a_i, $1 \leq i \leq n$, so that α is not k-sorted. For example, let $a_1 = 2$; $a_i = 1$, for $2 \leq i \leq i+k$; $a_i = 2$, for $k+2 \leq i \leq 2k+1$; and $a_{2k+2} = 1$; then $(a_1, \ldots, a_{2k+2})$ satisfies the two conditions of the theorem. □

Example 1. Suppose that every 5-block of $\alpha = (a_1, \ldots, a_7)$ is 2-sorted. Then, the partial order it specifies is depicted in Figure 1. If $\alpha = (6,2,3,7,8,5,9)$, this sequence is consistent with the partial order shown in Figure 1, but it is not 2-sorted.

3 The k-Bubble Sort

As is well known any sequence of length n(an *n-sequence* for short) can be sorted by at most $n-1$ bubbling passes, where the i-th bubbling pass is described as follows:

for $j := 1$ **to** $n - i$ **do**
  if $\alpha_j > \alpha_{j+1}$ **then** *exchange*(α_j, α_{j+1})

Note that α_j is synonymous with a_j and, in the following text, $\alpha_{i..j}$ denotes the subsequence $(a_i, \ldots, a_j)$.

The bubbling operation compares two adjacent items and exchanges them if they are not in sorted order. In other words, bubbling 0-sorts 2-blocks. We now extend the bubbling operation so that it k-sorts a $(2k+2)$-block. This operation is called *k-bubbling*. Obviously, ordinary bubbling is 0-bubbling. We give an algorithm, *k-bubble sort*, that k-sorts a given sequence by means of k-bubbling. Note that, in general, there are many k-sorted sequences corresponding to a $(2k+2)$-block and, for that matter, many k-sorted sequences corresponding to any given sequence. Uniqueness is achieved only when $k = 0$. Therefore, k-bubbling transforms a $(2k+2)$-block into one of its k-sorted sequences. However, for technical reasons we specify that k-bubbling be a particular transformation defined by the following procedure.

procedure k-$BUBBLE(\alpha, j, j+2k+1)$;
begin
  if $j = 1$ **then** k-*sort*$(\alpha, j, j+2k+1)$
    { To give any one of its k-sorted sequences}
  else α_{j+2k+1} is exchanged with the maximum
      among the items in $\alpha_{j..j+k}$ and α_{j+2k+1}
end

Using k-$BUBBLE$ a k-bubbling pass from the first $(2k+2)$-block to the last $(2k+2)$-block can be described as follows:

for $j := 1$ **to** $n - 2k - 1$ **do**
  k-$BUBBLE(\alpha, j, j+2k+1)$

At the beginning of the j-th $(j \geq 2)$ stage of a bubbling pass, the items in $\alpha_{j+k+1..j+2k}$ are k-sorted, since these items are k-sorted at the previous stage. Therefore, it is desirable that these items are not disturbed during the j-th stage. This is the reason why we specify k-bubbling as described in k-$BUBBLE$.

Lemma 3.1 *Suppose that* $\alpha = (a_1, \ldots, a_n)$ *and the following k-bubbling pass is executed:*

for $j := 1$ **to** $n - 2k - 1$ **do**
k-$BUBBLE(\alpha, j, j + 2k + 1)$

Then, for all i such that $n - k \leq i < n$, α_i is greater than or equal to any item in $\alpha_{1..i-k-1}$; that is, the last $k + 1$ items are k-sorted with respect to α.

Proof: The proof is by induction on the leftmost position j of the $2k + 2$ positions currently being processed in the k-bubbling pass. These $2k + 2$ positions are called the *window* of the k-bubbling pass.

Basis: $j = 1$. At the end of the first stage of the k-bubbling pass the last $k + 1$ items, in $\alpha_{1..2k+2}$, are obviously k-sorted.

Induction step: $j > 1$. Suppose that at the end of the j-th stage of the k-bubbling pass, the last $k+1$ items in the window are k-sorted with respect to $\alpha_{1..j+2k+1}$. During the $j + 1$st stage, the items in $\alpha_{j+k+2..j+2k+1}$ remain unchanged and any item in $\alpha_{1..j+k+1}$ cannot be greater than the items in the same position at the j-th stage. Hence, during the $j+1$st stage, the items in $\alpha_{j+k+2..j+2k+1}$ are k-sorted with respect to $\alpha_{1..j+2k+2}$. From the induction hypothesis, at the beginning of the $j + 1$st stage α_{j+k+1} is no smaller than any item in $\alpha_{1..j}$. Furthermore, α_{j+2k+2} at the end of the $j + 1$st stage is the maximum of the items in $\alpha_{j+1..j+k+1}$ and α_{j+2k+2}. Hence, α_{j+2k+2} is no smaller than any item in $\alpha_{1..j+k+1}$ at the end of the $j+1$st stage. Thus, the items in $\alpha_{j+k+2..j+2k+2}$ are k-sorted at the end of the $j + 1$st stage. (The situation for $k = 2$ is depicted in Figure 2.) Therefore, at the end of the k-bubbling pass, the last $k + 1$ items in α are k-sorted with respect to the whole sequence. □

From Lemma 3.1 we can design a k-bubble sorting algorithm as follows. The k-bubble sort consists of $\lceil (n - k - 1)/(k + 1) \rceil k$-bubbling passes. The length of the first bubbling pass is $n-2k-1$. The length of each subsequent bubbling pass is $k+1$ shorter than the previous bubbling pass. The k-bubble sort is a natural extension of bubble sort, and is also called the *roughly bubble sort*. The algorithm can be described as:

procedure $RBUBBLE(\alpha, 1, n, k)$;
begin
  for $i := 1$ **to** $\lceil (n - k - 1)/(k + 1) \rceil$ **do**
    for $j := 1$ **to** $n - k - (k + 1)i$ **do**
      k-$BUBBLE(\alpha, j, j + 2k + 1)$
    If $n - k - (k + 1)i < 1$ then k-$BUBBLE$ is executed once
    in the second **for** loop
end

Lemma 3.2 *α is k-sorted by $RBUBBLE(\alpha, 1, n, k)$.*

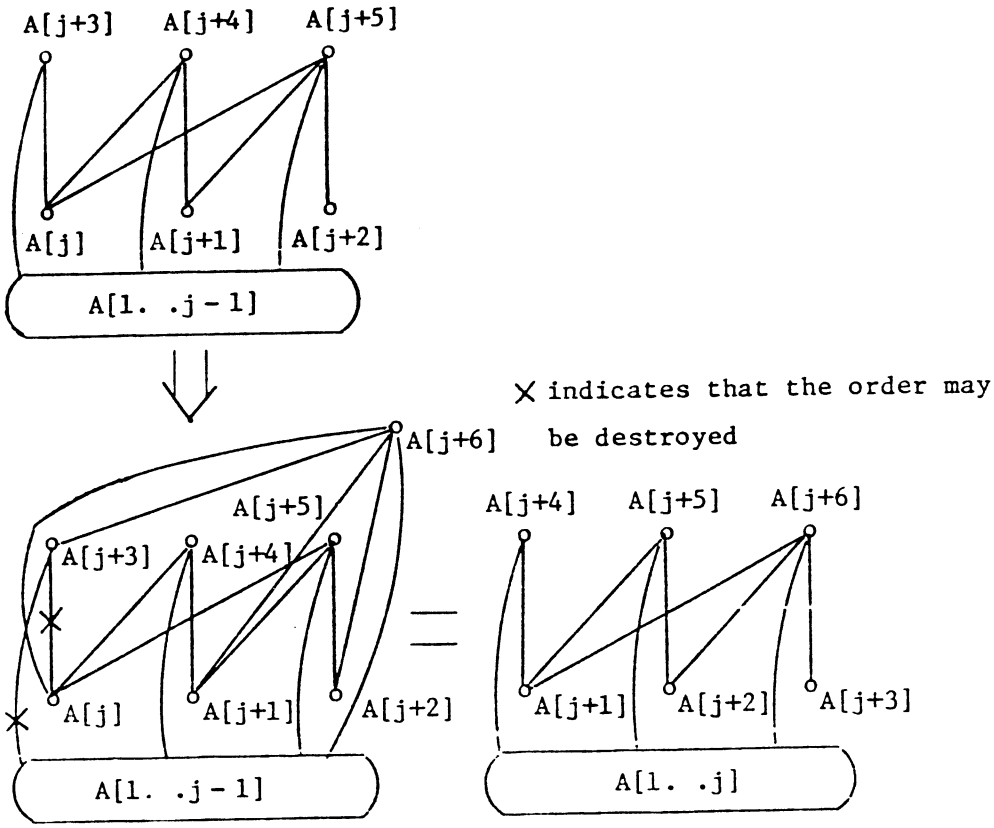

Figure 2: The change of a partial order by a one-position shift of the window in a 2-bubbling pass.

Proof: Let $P(i)$ be the following assertion

> At the end of the i-th bubbling pass in the computation of $RBUBBLE(\alpha, 1, n, k)$, the last $(k+1)i$ items are k-sorted with respect to the whole sequence.

If $(k+1)i > n$, the last $(k+1)i$ items in $P(i)$ should be read as the whole sequence. We prove that P(i) is true for all i, $1 \leq i \leq \lceil (n-k-1)/(k+1) \rceil$, by an induction on i.

Basis: $i = 1$. By Lemma 3.1, P(1) is true.

Induction step: $i > 1$. Suppose that P(i) holds and $i < \lceil (n-k-1)/(k+1) \rceil$. From the definition of k-$BUBBLE$, during the $i+1$st bubbling pass any item in $\alpha_{n-(k+1)(i+1)+1..n-(k+1)i}$ does not move unless it is smaller than an item in $\alpha_{1..n-(k+1)(i+1)}$. Therefore, the items in $\alpha_{n-(k+1)i+1..n}$ remain k-sorted throughout the $i+1$st bubbling pass. By Lemma 3.1, at the end of the $i+1$st bubbling pass the items in $\alpha_{n-(k+1)(i+1)+1..n-(k+1)i}$ are k-sorted with respect to $\alpha_{1..n-(k+1)i}$. Hence, at the end of the $i+1$st bubbling pass the last $(k+1)(i+1)$ items in $\alpha_{1..n}$ are k-sorted. Thus, the lemma holds. □

We first evaluate the computing time of $RBUBBLE$ as the number of calls of k-$BUBBLE$. By a simple calculation we obtain:

Lemma 3.3 *The number of calls of k-$BUBBLE$ to k-sort $\alpha = (a_1, \ldots, a_n)$ by $RBUBBLE$ is $\lfloor (n-k-1)/(k+1) \rfloor (n-2k+r)/2 + \beta(n)$, where $r = n$ modulo $(k+1)$, and $\beta(n) = 0$ if $r = 0$, otherwise $\beta(n) = 1$.*

Corollary 3.4 *When n is a multiple of $k+1$, the number of calls of k-$BUBBLE$ to k-sort $\alpha = (a_1, \ldots, a_n)$ by $RBUBBLE$ is $(n-k-1)(n-2k)/(2k+2)$.*

We next evaluate the computing time of $RBUBBLE$ in terms of the number of comparison-exchange operations instead of the number of calls of k-$BUBBLE$. Consider the sorting of a reverse ordered sequence of length n by bubble sorting. The sequence is initially $(n-1)$-sorted. Each bubbling pass reduces the unsorted size of the sequence by one. Therefore, in the worst case $n-k-1$ bubbling passes are required to obtain a k-sorted sequence from a given n-sequence. Thus, this straightforward method for k-sorting takes $(n-k-1)(n+k)/2$ comparison-exchanges. To evaluate the number of comparison-exchanges of $RBUBBLE$, we need to specify the details of k-$BUBBLE$. When $j = 1$, k-$BUBBLE(\alpha, j, j+2k+1)$ can be executed as follows:

```
for i := 1 to k + 1 do
    for s := 1 to i do
        if α_{k+1+i} < α_s then exchange(α_{k+1+i}, α_s)
```

When $j \geq 2$, the sequence is k-sorted by $k+1$ comparison-exchanges as follows:

for $i := 1$ **to** $k+1$ **do**
  if $\alpha_{j+2k+1} < \alpha_{j+i-1}$ **then** *exchange*$(\alpha_{j+2k+1}, \alpha_{j+i-1})$

Hereafter, k-*BUBBLE* means the computation specified by the above algorithm. From Lemma 3.3 the next theorem is immediate.

Theorem 3.5 *The number of comparison-exchanges to k-sort an n-sequence by RBUBBLE is $\gamma(k+1) + k(k+1)\lceil (n-k-1)/(k+1)\rceil/2$, where γ is the number of calls of k-BUBBLE given in Lemma 3.3 (that is, $\gamma = \lfloor (n-k-1)/(k+1)\rfloor (n-2k+r)/2 + \beta(n))$.*

Corollary 3.6 *When n is a multiple of $k+1$, the number of comparison-exchanges to k-sort an n-sequence by RBUBBLE is $(n-k-1)(n-k)/2$.*

From Corollary 3.6 we can say that *RBUBBLE* is faster than the straightforward method (that is, k-sorting by bubble sorting) by $k(n-k-1)$ comparison-exchanges. This is a pleasant result. For large k, if we use an $O(k \log k)$ sorting method for computing k-*BUBBLE* the computing time given in Lemma 3.3 can be marginally improved. However, a faster and optimal k-sorting algorithm is to be found in [3].

4 Sorting k-Sorted Sequences

In this section we study how to sort k-sorted sequences and how to merge two k-sorted sequences into a sorted sequence. For a k-sorted sequence $\alpha = (a_1, \ldots, a_n)$, for any i, $k+2 \leq i \leq n$, we have $a_i \geq a_j$ if $i \geq j+k+1$. Therefore, the following scheme can be used to sort a k-sorted sequence:

1. Let $\alpha(1) = \{a_1, \ldots, a_{k+1}\}$ and let $a_{s(1)}$ be a smallest item in $\alpha(1)$.
2. Suppose that for each t, $1 \leq t \leq i$, $a_{s(t)}$ has been found. Let $\alpha(i+1) = \alpha(i) - \{a_{s(i)}\} \cup \{a_{i+k+1}\}$, and let $a_{s(i+1)}$ be a smallest item in $\alpha(i+1)$. (If $i+k+1 > n$, $\{a_{i+k+1}\}$ should be read as the empty set).

Theorem 4.1 *For the above scheme, for all i, $1 \leq i \leq n$, $a_{s(i)}$ is an i-th smallest item in α.*

Proof: We prove the theorem by an induction on i. Since $a_{s(1)}$ is the smallest item in $\{a_1, \ldots, a_{k+1}\}$ and $(a_1, \ldots, a_n)$ is k-sorted, any item in $\{a_1, \ldots, a_n\}$ is no smaller than $a_{s(1)}$. Suppose that $a_{s(j)}$ is the j-th smallest item in α, for $j = 1, \ldots, i$. At the $i+1$st stage of the computation $a_{s(i+1)}$ is a smallest item in $\alpha(i+1)$. At least one item in $\{a_1, \ldots, a_{i+1}\}$ is in $\alpha(i+1)$, and

any item in $\{a_{i+k+1}, \ldots, a_n\}$ is no smaller than any item in $\{a_1, \ldots, a_{i+1}\}$. Hence, $a_{s(i+1)}$ is a smallest item in $\alpha(i+1) \cup \{a_{i+k+1}, \ldots, a_n\}$. Thus, $a_{s(i+1)}$ is an $i+1$st smallest item in α. □

A heap is known as a suitable data structure for a priority queue. In the following algorithm, based on the above scheme, we use a heap as a data structure both for finding $a_{s(i)}$ in $\alpha(i)$ and for constructing $\alpha(i+1)$.

```
procedure RHEAPSORT(α, 1, n, k);
begin
    Construct a heap for {a_1, ..., a_{k+1}};
    for i := 1 to n do
    begin
        choose the item at the root as the i-th smallest one;
        if i + k + 2 ≤ n then
            insert a_{i+k+2} at the root and trickle down
            until the heap condition is satisfied
        else
            move the last item of the heap to the root
            and trickle down until the heap condition is satisfied
    end
end
```

If $k = n-1$, $RHEAPSORT$ is exactly the same as the ordinary heap sort. The number of comparison-exchanges to construct a heap of size $k+1$ is $2(k+1) - 2\log_2(k+2)$, if $k = 2^t - 2$ for some t, while for arbitrary k it is bounded by $2(k+1)$. It takes at most $2\lfloor \log_2(k+1) \rfloor$ comparison-exchanges to update the heap. Thus, the computing time of $RHEAPSORT$ can be estimated as:

Theorem 4.2 *The number of comparison-exchanges needed by* $RHEAPSORT$ $(\alpha, 1, n, k)$ *is bounded by* $2(k+1) + 2(n-1)\lfloor \log_2(k+1) \rfloor$.

As is well known the optimal number of comparison-exchanges for 0-sorting is $\Theta(n \log n)$ [1,9]. From this fact and Theorem 4.2, the next corollary is immediate.

Corollary 4.3 *Let* $k = f(n)$. *If* $\lim_{n\to\infty}(\log f(n))/\log n = 0$, *then the optimal number of comparison-exchanges for k-sorting n items is* $\Theta(n \log n)$.

It is also well known that the optimal number of comparison-exchanges for finding the median of n items is $\Theta(n)$ [1,9], hence, we have:

Theorem 4.4 *Let k be a linear function of n such that its coefficient is less than 1 and greater than 0. Then the optimal number of comparison-exchanges for k-sorting n items is* $\Theta(n)$.

Proof: Suppose $k = rn$, where $0 < r < 1$. We consider the following algorithm for k-sorting $\alpha = (a_1, \ldots, a_n)$. Let t be a natural number such that $1/2^t < r$. We first find the median of α, and then partition α into two parts so that one part consists of items smaller than the median and the other consists of items equal to or greater than the median. We recursively apply the same procedure to the partitioned parts until the size of each partitioned part becomes smaller than $n/2^t$. Since we can find the median of a set in a linear number of comparison-exchanges and t is a constant, the number of comparison-exchanges of the algorithm is linear in n.

We next prove that a linear function of n is a lower bound on the number of comparison-exchanges of k-sorting algorithms. For any k-sorting algorithm, each of the items at positions $rn+2$ to n at the end of the computation has been compared with some item at least once during the computation. This fact is immediate from the following observation. If an item in this range has not been compared with any item during the computation, it may be the smallest item, in which case, the final sequence is not k-sorted. Therefore, any k-sorting algorithm needs $\Omega(n)$ comparison-exchanges. □

The technique used in $RHEAPSORT$ can be used to to merge two k-sorted sequences into a sorted sequence. We construct two heaps of size $k+1$ for this purpose. The algorithm is described by the following procedure. In the procedure we assume that we have two k-sorted sequences $\alpha = (a_1, \ldots, a_n)$ and $\beta = (b_1, \ldots, b_m)$.

```
procedure RMERGE(α, 1, n, β, 1, m, k);
begin
    construct a heap H_A for {a_1, ..., a_{k+1}}
    and a heap H_B for {b_1, ..., b_{k+1}};
    for i := 1 to m + n do
    begin
        choose the minimum of the items at the roots
        of H_A and H_B as the i-th smallest item;
            if the minimum item is chosen from H_A then
                the first unprocessed item in α is inserted
                at the root of H_A and H_A is updated
            else the first unprocessed item in β is inserted
                at the root of H_B and H_B is updated
        {We update H_A or H_B in the same way as in RHEAPSORT.
        For example, if the items in α are exhausted, the
        last item in H_A is moved to the root to update H_A.
        If H_A is empty, an item is chosen from the
        root of H_B and H_B is updated.
        H_B is updated in the same way as H_A.}
```

```
    end
end
```

The computing time of $RMERGE$ can be estimated in the same way as that of $RHEAPSORT$.

Theorem 4.5 *The number of comparison-exchanges used in the computation by* $RMERGE(\alpha, 1, n, \beta, 1, m, k)$ *is bounded by* $4(k+1) + 2(m+n-2)\lfloor \log_2(k+1) \rfloor + \min(m, n)$.

5 The Number of k-Sorted Sequences

Throughout this section, we assume that all items of a given sequence are distinct. We denote the number of k-sorted sequences of an n- sequence by $R_k(n)$ (that is, $R_k(n)$ is the number of permutations of $(1, \ldots, n)$ that are k-sorted). We estimate $R_k(n)$ and derive a lower bound on the number of comparison-exchanges for sorting a k-sorted sequence. We first consider the case for $k = 1$. Suppose that $\alpha = (a_1, \ldots, a_n)$ is 1-sorted. The partial order specified by the sequence is shown in Figure 3. The order of a_1 and a_2 is either $a_1 < a_2$ or $a_1 > a_2$. The number of 1-sorted sequences such that the first item is smaller than the second item (that is, the first item is the smallest) is $R_1(n-1)$, and the number of 1-sorted sequences such that the first item is greater than the second item is $R_1(n-2)$. Therefore, we have the recurrence equation $R_1(n) = R_1(n-1) + R_1(n-2)$. Since $R_1(1) = 1$ and $R_1(2) = 2$, $R_1(n)$ is the n-th Fibonacci number. The asymptotical value of $R_1(n)/R_1(n-1)$ is the positive root of $x^2 - x - 1 = 0$. Hence, the next theorem is obtained.

Theorem 5.1 $R_1(n)$ *is the n-th Fibonacci number. For a sufficiently large* n, $R_1(n)/R_1(n-1) \approx (1+\sqrt{5})/2 (= 1.618\cdots)$.

From the above theorem and a decision tree argument [1] the next theorem is obtained.

Theorem 5.2 *A lower bound on the number of comparisons needed in the average case for sorting a 1-sorted n-sequence is* $n \log_2 1.618 - 1 \approx 0.69428n - 1$.

The bound given in the above theorem is also a lower bound in the worst case. However, we can derive a better lower bound in the worst case.

Theorem 5.3 *Any algorithm for sorting 1-sorted n-sequences needs* $n-1$ *comparisons in the worst case.*

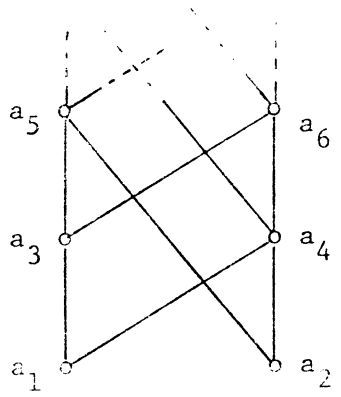

Figure 3: The partial order specified by the 1-sorted sequence $(a_1, a_2, \cdots)$.

Proof: Let $\alpha = (a_1, \ldots, a_n)$ be a 1-sorted sequence. Any comparison of a_i and a_j such that $i \geq j+2$ or $i \leq j-2$ does not contribute to deciding the order of any items of unknown order. Consider any algorithm for sorting 1-sorted sequences. Suppose that a_i and a_{i+1} are compared in the computation of the algorithm. If $a_i \leq a_{i+1}$, this comparison does not contribute to deciding the order of any other items apart from a_i and a_{i+1}. If at each comparison of a pair of successive items the first is no greater than the second, the number of pairs of successive items of unknown order is reduced by one. Since α has initially $n-1$ successive pairs of items of unknown order, the algorithm needs at least $n-1$ comparisons to sort the sequence. □

For small k, *RHEAPSORT* is not particularly efficient. The straight insertion method [4,9] may be a suitable method for sorting k-sorted sequences, if k is small. It takes at most $k(n-1)$ comparisons. However, the straight insertion method needs $O(k^2 n)$ item movements if it is implemented in an array. The following algorithm sorts any 1-sorted n-sequence in at most $n-1$ comparisons.

```
procedure ONESORT(α, 1, n);
begin
    i := 1;
    while i ≤ n − 1 do
        if α_i ≤ α_{i+1} then i := i + 1
        else begin
            exchange(α_i, α_{i+1});
            i := i + 2
        end
```

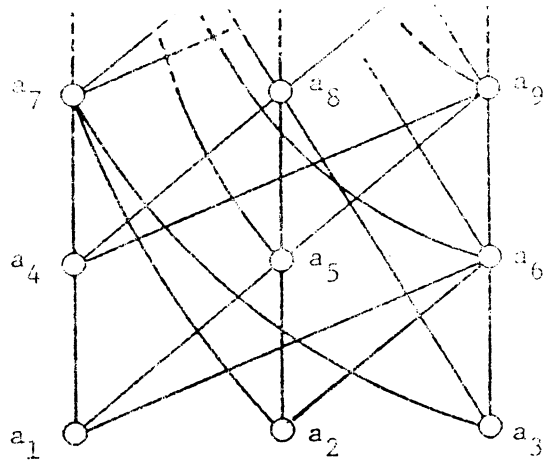

Figure 4: The partial order of the 2-sorted sequence $(a_1, a_2, \cdots)$.

end

From Theorem 5.3 and the above algorithm we have:

Theorem 5.4 *The worst case optimal number of comparisons to sort 1-sorted sequences of length n is $n-1$.*

Theorem 5.5 *For sufficiently large n, the average number of comparisons to sort 1-sorted sequences of length n by $ONESORT$ is $\approx 0.7236n$.*

Proof: After each comparison of two items in $ONESORT$, i is incremented by 1 or 2 in the **while** loop. The numbers of 1-sorted sequences $(a_i, \ldots, a_n)$ such that $a_i \leq a_{i+1}$ or such that $a_i > a_{i+1}$ are $R_1(n-i)$ and $R_1(n-i-1)$, respectively. Since $R_1(n-i+1) = R_1(n-i) + R_1(n-i-1)$, the expected increment is $R_1(n-i)/R_1(n-i+1) + 2R_1(n-i-1)/R_1(n-i+1)$. Thus, from Theorem 5.1, for sufficiently large $n-i$, the expected increment is $1/r + 2/r^2$, where $r = (1+\sqrt{5})/2$. Hence, the expected number of comparisons is $\approx n/(1/r + 2/r^2) \approx 0.7236n$. □

As shown in the above theorem, the average number of comparisons taken by $ONESORT(\alpha)$ is close to the average case lower bound $0.69438n$ given in Theorem 5.2.

A recurrence equation for $R_2(n)$ can be derived by the following case analysis on the partial order specified by a 2-sorted sequence $\alpha = (a_1, a_2, \cdots)$. The partial order of α is shown in Figure 4.

1. The number of 2-sorted n-sequences such that $a_1 < a_2, a_3$ is $R_2(n-1)$.

2. The number of 2-sorted n-sequences such that $a_2 < a_1 < a_3$ is $R_2(n-2)$.

3. The number of 2-sorted n-sequences such that $a_2, a_3 < a_1$ is $2R_2(n-3)$.

4. The number of 2-sorted sequences n-sequences such that $a_3 < a_1 < a_2$ and $a_2 < a_4$ is $R_2(n-3)$.

5. The number of 2-sorted n-sequences such that $a_3 < a_1 < a_2$ and $a_4 < a_2$ is $R_2(n-4)$.

From the above case analysis we have the following recurrence equation for $R_2(n)$:

$$R_2(n) = R_2(n-1) + R_2(n-2) + 3R_2(n-3) + R_2(n-4);$$

$$R_2(1) = 1; R_2(2) = 2; R_2(3) = 6; R_2(4) = 12$$

Solving the recurrence equation we obtain:

Theorem 5.6 *For sufficiently large n, $R_2(n)/R_2(n-1) \approx r \approx 2.1833$, where r is the positive root of $x^4 - x^3 - x^2 - 3x - 1 = 0$.*

By a decision tree argument we have:

Theorem 5.7 *A lower bound on the number of comparisons both in the worst case and in the average case for sorting 2-sorted sequences of length n is $n \log_2 2.1833 - 2 \approx 1.1265n - 2$.*

For larger values of k, the case analysis to obtain a recurrence equation for $R_k(n)$ seems to be complex. We, therefore, try to derive a recurrence inequality for $R_k(n)$. We focus our attention on the first $k+1$ items of a k-sorted sequence and estimate $R_k(n)$ by the following case analysis:

1. The number of k-sorted n-sequences such that $a_1 < a_2, a_3, \ldots, a_{k+1}$ is $R_k(n-1)$.

2. The number of k-sorted n-sequences such that $a_2, \ldots, a_{t+1} < a_1 < a_{t+2}, \ldots, a_{k+1}$ is $t!R_k(n-t-1)$, where $1 \leq t \leq k$.

3. The number of k-sorted n-sequences such that $a_1, \ldots, a_{k+1} < a_{k+2}, \ldots, a_n$ is $(k+1)!R_k(n-k-1)$. $k!R_k(n-k-1)$ sequences and $\Sigma_{t=1}^{k} t!R_k(n-k-1)$ sequences among the above $(k-1)!R_k(n-k-1)$ sequences have been already counted in (1) and (2) above, respectively. Therefore, the number of k-sorted n-sequences such that they are neither in (1) nor in (2) above is more than $((k+1)! - k! - \Sigma_{t=1}^{k} t!)R_k(n-k-1)$.

From the above case analysis we have the following recurrence inequality:

$$R_k(n) > R_k(n-1)+\Sigma_{t=1}^{k}(t!R_k(n-t-1))+((k+1)!-k!-\Sigma_{t=1}^{k}t!)R_k(n-k-1)$$

$$R_k(i) = i!;\ i = 1,\ldots,k+1$$

Therefore, for suffciently large n, $R_k(n)/R_k(n-1) > r$, where r is the positive root of

$$x^{k+1} - (x^k + \Sigma_{t=1}^{k-1}t!x^{k-t} + (k+1)! - k! - \Sigma_{t=1}^{k-1}t!) = 0.$$

By a decision tree argument, $n\log_2 r$ is a lower bound on the number of comparisons both in the worst case and in the average case for sorting k-sorted sequences of length n, where n is sufficiently large. The positive roots r of the above equations for $k = 3,\ldots,10$ are given in Table 1. For example, $2.1183n$ is an average case lower bound on the number of comparisons needed to sort an 8-sorted sequence of length n.

k	lower bound on $R_k(n)/R_k(n-1)$
3	2.57231
4	2.95041
5	3.30127
6	3.64575
7	3.99178
8	4.34201
9	4.69658
10	5.05497

Table 1: Lower bounds on $R_k(n)/R_k(n-1)$.

6 Concluding Remarks

We have defined the notion of roughly sortedness and studied several problems related to roughly sorting. For $k = 1$ and 2, the numbers of k-sorted sequences have been estimated. From these estimates we have derived lower bounds on the number of comparisons needed to sort 1-sorted sequences and 2-sorted sequences. However, for larger k we have only been able to underestimate such a number. If we could estimate these numbers, a better lower bound on the number of comparisons needed to sort a k-sorted sequence for larger k could be derived. A number of interesting problems related to roughly sorting and roughly sorted sequences can be considered.

We have designed the k-bubble sort as a generalization of bubble sort. We are also interested in designing k-sorting versions of other well known sorting algorithms, and an efficient algorithm for merging two k_1-sorted sequences into a k_2-sorted sequence. Another interesting problem is the trade-off between the computing time and (k_1, k_2) of the above merging algorithm. All problems discussed in this paper are in the case of serial computing. Parallel algorithms related to roughly sorting are also interesting and worthy of further investigation.

Acknowledgement

The first author wishes to thank Professor Maarten H. van Emden for his useful discussions and encouragement during his stay in Waterloo.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Co., Reading, Mass., 1974.

[2] C.R. Cook and D.J. Kim. Best sorting algorithms for nearly sorted lists. *Communications of the ACM*, 23:620–624, 1980.

[3] V. Estivill-Castro and D. Wood. *A New Measure of Presorting.* Technical Report 87-CS-??, Department of Computer Science, University of Waterloo, 1987.

[4] D.E. Knuth. *The Art of Computer Programming, Vol.3: Sorting and Searching.* Addison-Wesley Publishing Co., Reading, Mass., 1973.

[5] H. Mannila. Measures of presortedness and optimal sorting algorithms. *IEEE Transactions on Computers*, C-34:318–325, 1985.

[6] K. Mehlhorn. Sorting presorted files. In *4th GI Conference on Theory of Computer Science*, pages 199–212, Springer-Verlag, New York, 1979.

[7] K. Sado and Y. Igarashi. *A Divide-and-Conquer Method of The Parallel Sort.* Technical Report AL84-68, IECEJ, 1984.

[8] K. Sado and Y. Igarashi. *A Fast Parallel Pseudo-Merge Sort Algorithm.* Technical Report AL85-16, IECEJ, 1985.

[9] N. Wirth. *Algorithms + Data Structures = Programs.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.