Two probabilistic results on merging

Wenceslas Fernandez de la Vega* Sampath Kannan†
Miklos Santha‡

Abstract

This paper contains two probabilistic results about merging two sorted lists of sizes n and m with $n < m$. We design a probabilistic algorithm which in the worst case is significantly faster than any deterministic one in the range $1.618 < m/n \leq 3$. We extend it into a simple general algorithm which performs well for any ratio m/n. In particular, for $m/n > 1.618$ it is significantly faster than binary merge. We also prove an average case lower bound for a wide class of merging algorithms, when $1 < m/n < \sqrt{2} + 1$.

1 Introduction

Merging is one of the basic problems in theoretical computer science. Given two sorted lists $A = \{a_1 < \ldots < a_n\}$ and $B = \{b_1 < \ldots < b_m\}$, the task consists of sorting the union of the two lists. We assume that the $n + m$ elements are distinct and $n \leq m$. The merging is performed by pairwise comparisons between items in A and items in B. The measure of complexity of a merging algorithm is the number of comparisons made by the algorithm, and the complexity of the merging problem is the complexity of the best merging algorithm. As usual, we can speak of worst case and average case complexity.

The worst case complexity of the merging problem is quite well studied. Let $C(n, m)$ denote this complexity with input lists of size n and m. When $n = 1$, merging degenerates into binary search whose complexity is $\lceil \log_2(m + 1) \rceil$. The case $n = 2$ was completely solved by Graham [7] and Hwang and Lin [4], their result is

$$C(2, m) = \lceil \log_2 7(m + 1)/12 \rceil + \lceil \log_2 14(m + 1)/17 \rceil.$$

The exact complexity is also known when the sizes of the two lists are not too far away. For $n \leq m \leq \lfloor 3n/2 \rfloor$ we have

$$C(n, m) = n + m - 1,$$

and the optimal algorithm is the well known tape-merge (two-way merge). This result was obtained by Stockmeyer and Yao [9] and by Christen [2], after it was observed by several

*Université Paris-Sud, 91405 Orsay, France
†University of California, Berkeley CA 94720, Supported by NSF Grant CCR85-13926
‡Université Paris-Sud, 91405 Orsay, France

people for $n \leq m \leq n+4$. The best known algorithm — binary merge — which gives satisfying result for any values of n and m is due to Hwang and Lin [5]. Let $\mathcal{N}$ denote the set of non-negative integers. Set $m/n = 2^t x$, where $t \in \mathcal{N}$ and $1 < x \leq 2$ are uniquely determined, and let $BM(n,m)$ denote the complexity of binary merge. Then

$$BM(n,m) = \lfloor (t+x+1)n \rfloor - 1,$$

and Hwang and Lin have also shown that

$$BM(n,m) \leq L(n,m) + n,$$

where $L(n,m) = \log_2 \binom{n+m}{n}$ is the lower bound coming from information theory. This means that if the ratio m/n is high, then the relative overwork done by binary merge is certainly small. As the relative overwork might be significant for small ratios of m/n, it is important to look for improvements in this case.

Several algorithms were proposed which in some range for m/n perform better than binary merge. Let us say that merging algorithm A_1 with running time $T_1(n,m)$ is *significantly* faster for some fixed ratio m/n than merging algorithm A_2 with running time $T_2(n,m)$, if $T_2(n,m) - T_1(n,m) = \Omega(n)$. The merging algorithm of Hwang and Deutsch [6] is better than binary merge for small values of n, but is not significantly faster. The first significant improvement over binary merge was proposed by Manacher [8], he improved it for $m \geq 8n$ by $31n/336$ comparisons. It was further improved by Christen [1] who designed an ingenious but quite involved merging algorithm. It is better than binary merge if $m > 3n$, it uses at least $n/4$ comparisons less if $m \geq 4n$, and asymptotically it uses at least $n/3 - o(n)$ comparisons less when m/n goes to infinity. On the other hand, this algorithm is worst than binary merge when $m < 3n$.

In the first part of this paper we propose a probabilistic algorithm for merging. The algorithm will at some points flip a (biased) coin, and its next step will depend on the result of the coin toss. The algorithm is quite simple, and works well for every values of n and m. It is significantly faster then binary merge for $m/n > (\sqrt{5}+1)/2 \approx 1.618$. To the best of our knowledge binary merge is the best algorithm known for $m/n \leq 3$, thus our probabilistic algorithm is significantly faster than any deterministic one in this range.

In the second part of the paper we prove a non-trivial average case lower bound for a wide class of merging algorithms. We say that a merging algorithm is *insertive* if for each element of the smaller list the comparisons which involve it are all made one after the other. In other words in the binary decision tree associated with the algorithm, along any path, for every element a of the smaller list, the nodes which represent comparisons involving the element a appear in consecutive positions. Apart from this constraint the decision tree can be arbitrary. Many merging algorithms such as repeated binary search, tape merge and binary merge are insertive. We will prove that if $(1+\eta) \leq m/n \leq (\sqrt{2}+1-\eta)$ for some constant $\eta > 0$, then every insertive merging algorithm makes on the average at least a constant factor more comparisons than the information theoretical lower bound.

The rest of the paper is organized as follows: In Section 2.1 we describe our algorithm. The analysis of its complexity will be done in Section 2.2 and we also prove there that the particular bias of the coin in the algorithm was optimally chosen. In Section 3 we prove the average case lower bound result for insertive algorithms. Finally, in Section 4

we mention some open problems. Throughout the paper the logarithm function in base 2 will be denoted log.

2 The probabilistic algorithm

2.1 Description of the algorithm

For this section let $s = (\sqrt{5}-1)/2 \approx 0.618$, and $r = (\sqrt{2}-1+\sqrt{2s})^2 \approx 1.659$, these number play a considerable role in the algorithm and in its analysis. The heart of the algorithm MERGE is the probabilistic procedure PROBMERGE which merges two already sorted lists, where the longer list contains more than $(1+s)$-times, but at most $2r$-times as many elements as the shorter one. The intuition underlying PROBMERGE is described below. We know that if $m \leq \lfloor 3n/2 \rfloor$ then the tape merge algorithm is best possible. The tape merge algorithm can be thought of as inserting the a's in order in list B. At each stage the next a to be inserted is compared with the first 'eligible' element in B. If however B is, for example, a list of length $2n$ then since on the average between every two consecutive a's there are two b's, it might be better to compare the next a to be inserted with the second eligible b. However a deterministic algorithm which tries to do this turns out to be no better than tape merge. Its worst case complexity is also $3n-1$. A natural idea is to try to compare the next a to be inserted with either the first or the second eligible element in B, the decision being made probabilistically. In fact this is our algorithm.

We will use sentinel elements in the algorithm. They make the description simpler for the price of a little loss of efficiency.

Procedure PROBMERGE

Input: $A = \{a_1 < \ldots < a_n\}$ and $B = \{b_1 < \ldots < b_m\}$, where $1+s < m/n \leq 2r$, and sentinel elements $b_0 = -\infty$, $b_{m+1} = b_{m+2} = \infty$.

Output: The merged list.

- $p := \begin{cases} s & \text{if } 1+s < m/n \leq 2+s \\ \sqrt{m/n}-1 & \text{if } 2+s < m/n \leq 2r \end{cases}$ $\left\{ \begin{array}{l} \textit{the choice of the} \\ \textit{probability value} \end{array} \right\}$
- $i := 1; j := 1$ $\{i$ indexes A, j indexes $B\}$
- **while** $i \leq n$ **do**
  - with probability $1-p$ compare a_i with b_j
    - **if** $a_i < b_j$ **then** $i := i+1$ $\{b_{j-1} < a_i < b_j\}$
    - **else** $j := j+1$
  - with probability p compare a_i with b_{j+1}
    - **if** $a_i < b_{j+1}$ **then**
      - compare a_i with b_j
      - **if** $a_i < b_j$ **then** $i := i+1$ $\{b_{j-1} < a_i < b_j\}$
      - **else** $i := i+1; j := j+1$ $\{b_j < a_i < b_{j+1}\}$

* **else** $j := j + 2$

- **end**

The general algorithm MERGE uses PROBMERGE as a subroutine. Given lists A and B of size n and $m > (1+s)n$ respectively, the algorithm calls directly PROBMERGE if $m \leq rn$. Otherwise it picks a uniformly spaced sublist C of B of cardinality between rn and $2rn$. More precisely, the spacing between the elements of C is 2^t for t a non-negative integer. The algorithm first merges A with C probabilistically. This determines for each a in A a list of $2^t - 1$ b's in which a should be inserted. The insertion is done deterministically by binary search and takes t steps. Let us observe that MERGE actually coincides with the procedure PROBMERGE while $m \leq 2rn$.

Algorithm MERGE

Input: $A = \{a_1 < \ldots < a_n\}$ and $B = \{b_1 < \ldots < b_m\}$, where $(1+s) < m/n$. For $r < m/n$ set $m = 2^t xn$, $t \in \mathcal{N}$, $r < x \leq 2r$.

Output: The merged list.

1. If $m/n \leq r$ then PROBMERGE(A, B).
2. Let $C = \{c_1 < \ldots < c_{\lceil xn \rceil}\}$ be the sublist of B, where $c_k = b_{(k-1)2^t+1}$ for $k = 1, \ldots, \lceil xn \rceil$, and define sentinel elements $c_0 = -\infty$, $c_{\lceil xn \rceil + 1} = \infty$.
3. PROBMERGE(A, C).
4. For $i = 1, \ldots, n$ let $0 \leq j_i \leq \lceil xn \rceil$ be such that $c_{j_i} < a_i < c_{j_i+1}$.
5. For $i = 1, \ldots, n$ insert a_i by binary search into the list $\{b_{(j_i-1)2^t+2} < \cdots < b_{j_i 2^t}\}$.

2.2 Analysis of the algorithm

We would like to analyse the expected number of comparisons made by the algorithm PROBMERGE. Let $T(n, m)$ be this number when the input of the procedure is two lists of size respectively n and m, $1 + s < m/n \leq 2r$, and let p be the probability value defined inside the procedure.

Theorem 1 *We have*

$$T(n,m) \leq \begin{cases} sm + (1+s)n & \text{if } 1+s < m/n \leq 2+s \\ 2\sqrt{mn} & \text{if } 2+s < m/n \leq 2r. \end{cases}$$

Proof The outcome of the procedure uniquely determines n non-negative integers $k_1, \ldots, k_n$, where k_i is the number of b's which are between a_{i-1} and a_i. As the list B contains m elements, $\sum_{i=1}^{n} k_i \leq m$. The procedure puts the a's into B one by one. Let f_k be the expected number of comparisons to put an a into B, when the number of b's between a

and its predecessor in A is k. f_k doesn't depend on the index of the element being inserted. Then

$$T(n,m) = \max_{\substack{k_1,\ldots,k_n \\ \sum_{i=1}^n k_i \le m}} \sum_{i=1}^n f_{k_i}.$$

Thus we are interested in the values f_k. If $k \ge 2$, then after the first comparison the element a jumps over two b's with probability p, and jumps over one b with probability $1-p$. This means that we get the following recurrence relation for f_k:

$$f_k = (1-p)f_{k-1} + pf_{k-2} + 1,$$

with the initial conditions

$$f_0 = 2p + 1(1-p) = p+1,$$
$$f_1 = 3p(1-p) + 2((1-p)^2 + p) = -p^2 + p + 2.$$

By standard technique, the solution of this linear recurrence is

$$f_k = \frac{k}{p+1} + \frac{p^3+p^2-p}{(p+1)^2}(-p)^k + \frac{2p^2+4p+1}{(p+1)^2}.$$

This tells us that

$$\sum_{i=1}^n f_{k_i} = \sum_{i=1}^n \frac{k_i}{p+1} + \frac{p^3+p^2-p}{(p+1)^2}\sum_{i=1}^n (-p)^{k_i} + \frac{2p^2+4p+1}{(p+1)^2}n.$$

The term $\sum_{i=1}^n \frac{k_i}{p+1}$ is always bounded from above by $\frac{m}{p+1}$. If $p = s$, we have $p^3+p^2-p = 0$, and the result follows. If $p > s$ (when $2+s < m/n$), we have $p^3+p^2-p > 0$, and $\sum_{i=1}^n (-p)^{k_i}$ is maximized by choosing $k_i = 0$. Simple arithmetic gives the result also in this case. □

The global algorithm MERGE calls PROBMERGE with two lists of size n and $\lceil nx \rceil$ and then makes n binary searches, each in a list of size $2^t - 1$. Applying Theorem 1, we immediately get the following result:

Theorem 2 *Let $E(n,m)$ denote the expected number of comparisons made by* MERGE. *For $m/n \le r$ set $m/n = x$, and for $m/n > r$ set $m/n = 2^t x$, $t \in \mathcal{N}$, $r < x \le 2r$. Then*

$$E(n,m) \le \begin{cases} (t+sx+1+s)n+1 & \text{if } r < x \le 2+s \\ (t+2\sqrt{x})n+1 & \text{if } 2+s < x \le 2r. \end{cases}$$

Corollary 1 *The algorithm* MERGE *is significantly faster than binary merge for any fixed ratio $m/n > 1+s$.*

Proof With the notation of Theorem 2 we have

$$BM(n,m) - E(n,m) \ge \begin{cases} (x-sx-s)n-3 & \text{if } (1+s) < x \le 2 \\ ((1-s)-x(s-1/2))n-3 & \text{if } 2 < x \le 2+s \\ (2-\sqrt{x})^2 n/2 - 3 & \text{if } 2+s < x \le 2r. \end{cases}$$

This difference is $\Omega(n)$. □

Let us point out an interesting special case of the above result. When $m = 2n$, PROBMERGE merges the two lists with less than $2.855n$ expected comparisons, whereas the best known deterministic algorithm [5] performs $3n - 2$ comparisons. A simple calculation also yields the following relation between the expected running time of MERGE and the information theoretic lower bound. The maximum difference is attained for values of $m/n \to \infty$ with $x = 2r$.

Corollary 2 $E(n,m) - L(n,m) \leq 0.471n$

We also claim that the probability value p was optimally chosen in the procedure PROBMERGE. Let $T_p(n,m)$ be the expected number of comparisons made by PROBMERGE when the probability $0 \leq p \leq 1$ is taken as a variable. Let us observe that we have just analyzed $T_s(n,m)$ and $T_{\sqrt{m/n}-1}(n,m)$ in Theorem 1. For the purpose of this analysis let us permit any input lists such that the ratio m/n is at least 1. The following theorem which we state here without proof can be obtained by doing some calculus.

Theorem 3 *For every large enough n and m, for every p we have*

$$T_p(n,m) \geq \begin{cases} T_0(n,m) & \text{if } 1 \leq m/n \leq 1+s, \\ T_s(n,m) & \text{if } 1+s \leq m/n \leq 2+s \\ T_{\sqrt{m/n}-1}(n,m) & \text{if } 2+s \leq m/n \leq 4, \\ T_1(n,m) & \text{if } 4 \leq m/n. \end{cases}$$

Thus while $m/n \leq (1+s)$, our probabilistic procedure gives out deterministic tape-merge as a special case. In the range $1+s \leq m/n \leq 2+s$ the best choice is the constant $p = s$. For $m/n > 2+s$, the best choice grows with m/n until 1, when the procedure (obviously) degenerates into a deterministic one. There is no reason to use PROBMERGE for large values of m/n, MERGE is significantly faster when $m/n > 2r$. In fact that is how r is chosen.

3 The lower bound

In this section we will prove the following theorem:

Theorem 4 *For every $\eta > 0$ there exists $\epsilon > 0$ such that for every large enough n and m with $(1+\eta)n \leq m \leq (\sqrt{2}+1-\eta)n$, every insertive merging algorithm makes at least $(1+\epsilon)\log L(n,m)$ comparisons on the average.*

Proof Here we will prove the theorem in the special case $m = 2n$ wich already contains the main ideas for the general result. The proof depends on the following Unbalance Lemma:

Unbalance Lemma *Let T denote a binary decision tree with k leaves, let the set of internal nodes be $\{D_1, \ldots, D_{k-1}\}$, and let d_j be the number of leaves of the subtree with root D_j. Let us suppose that for some $J \subseteq \{1,2,\ldots,k-1\}$, the subset of nodes $\{D_j : j \in J\}$ satisfies:*

i) There exists $\epsilon_1 > 0$ *such that* $\sum_{j \in J} d_j \geq \epsilon_1 k \log_2 k$.

ii) There exists $\epsilon_2 > 0$ *such that for every* $j \in J$, *the answer probabilities (fraction of leaves in left and right subtrees) at the node* D_j *lie outside the interval* $(1/2 - \epsilon_2, 1/2 + \epsilon_2)$.

Then, the average path length of T *is at least*

$$(1 + \epsilon_1 \epsilon_2^2) \log_2 k. \quad \square$$

Let us fix some insertive algorithm, and let T be the binary decision tree associated with this algorithm. Let the internal nodes of T be $D_1, \ldots, D_{k-1}$, where $k = \binom{n+m}{n}$. Since the average running time of the algorithm is the average path length of T, it will suffice to prove that the assumptions of the Unbalance Lemma are fulfilled for some subset of the nodes and for some proper ϵ_1 and ϵ_2. To begin, notice that there is a bijection between the outcomes of a merging algorithm and the set Z of words containing precisely n letters, a and m letters, b. For any word $w \in \{a, b\}^*$, let $|w|_a$ and $|w|_b$ denote respectively the number of occurrences of a and b in w. Let $c_3 > 0$ be an appropriate constant which will be specified later. For any $n \leq i \leq m$ and $w \in Z$, let $decomp(w, i)$ denote the decomposition $uvxy = w$, where the factors u, v, x, y have the following lengths:

- $|u| = i - 3c_3$,
- $|v| = |x| = 3c_3$,
- $|y| = 3n - i - 3c_3$.

Let us denote by Z_i the set of words $w \in Z$ such that $decomp(w, i)$ satisfies the following conditions:

1. $vx = (bba)^{c_3}(abb)^{c_3}$,
2. for every decomposition $y = y'y''$, where $|y'| = l$, and for every decomposition $u = u''u'$ where $|u'| = l$, we have

$$(2/3 - 1/30)l - c_2 \leq |y'|_b, |u'|_b \leq (2/3 + 1/30)l + c_2,$$

where $c_2 > 0$ is an appropriate constant also to be specified later. The value $1/30$ in the second condition is arbitrary, any constant $\theta > 0$ such that $2/3 - \theta > 1/2$ and $2/3 + \theta < 1/\sqrt{2}$ will suffice. We will define a subset of nodes satisfying the assumptions of the Unbalance Lemma using the sets Z_i. First we prove two claims about these sets.

Claim 1 *There exists a constant* $\epsilon_3 > 0$ *such that for all* $n \leq i \leq m$ *we have:*

$$|Z_i| \geq \epsilon_3 |Z| .$$

Proof Let w be a random element of Z, and let $n \leq i \leq m$. The first requirement is obviously satisfied by w with constant probability. We will show that under this hypothesis, the conditional probability that $|y'|_b$ falls outside the interval $(19l/30 - c_2, 7l/10 + c_2)$ is less then 0.48 . As the second requirement is symetrical in $|y'|_b$ and $|u'|_b$, this will imply the claim. We shall (rather crudely) bound this probability by the sum $\sum_{1 \leq l \leq |y|} s(l)$, where $s(l)$ denotes the probability that the left factor of y of length l violates the second requirement. Clearly $s(l) = q(l) + r(l)$, where $q(l)$ is the probability that the left factor of y of length l contains less than $19l/30 - c_2$ occurrences of b's, and $r(l)$ is the probability that the same left factor contains more than $7l/10 + c_2$ occurrences of b's. Hoeffding's bound [3] about sampling without replacement gives us for every l,

$$q(l), r(l) < 0.9992^l.$$

Let us choose c_1 such that $\sum_{l=c_1+1}^{\infty} 0.9992^l < 0.24$. We can take for example $c_1 = 10800$ (this is not the smallest possible choice). Now, in function of c_1 we can choose c_2 such that $q(l) = r(l) = 0$ whenever $l \leq c_1$. If we take $c_2 = 19c_1/30 = 6840$, then $19l/30 - c_2 \leq 0$, and clearly $q(l) = 0$. It can be checked that this choice is good also for $r(l)$. Hence

$$\sum_{1 \leq l \leq |y|} s(l) < 2 \sum_{l=10800+1}^{\infty} 0.9992^l < 0.48 . \quad \square$$

We are now ready to choose the constant c_3 in function of c_2 such that the following Claim becomes true. Let $c_3 = 10c_2 = 68400$.

Claim 2 *Let $w \in Z_i$ for some $n \leq i \leq m$, and let $uvxy = decomp(w,i)$. Let z be a non-empty prefix of xy (a suffix of uv) such that either the letter which follows (precedes) z in w is an a, or there is no more a in w after (before) z. Then we have*

$$19/30 \leq |z|_b/|z| \leq 7/10.$$

Proof We show the upper bound. If z is a prefix of x, than $|z|_b/|z| = 2/3$. Otherwise this fraction is maximized when the corresponding prefix of y contains the most possible b's. If this prefix is of length l, we have

$$|z|_b/|z| \leq \frac{2c_3 + 7l/10 + c_2}{3c_3 + l} = 7/10. \quad \square$$

We can now define a subset of the nodes of T which satisfies the conditions of the Unbalance Lemma. For every couple (w,i) such that $n \leq i \leq m$ and $w \in Z_i$, we first define an internal node $V(i,w)$ of T. Let $uvxy = decomp(w,i)$, and let $h = h(w,i)$ be such that a_h is the first instnce of a in x (thus $h-1$ is the number of instances of a in uv). If on the path of T corresponding to w the comparisons involving a_{h-1} preceede those involving a_h, then let $V(w,i)$ be the node of the decision tree at which the first comparison involving a_h takes place. Otherwise let $V(w,i)$ be the node at which the first comparison involving a_{h-1} takes place. Finally let

$$J = \{j : D_j = V(w,i) \text{ for some } w \in Z_i\}.$$

We now show that the two conditions of the Unbalance Lemma are met by the nodes whose indices are in J. First we will show that

$$\sum_{j\in J} d_j \geq \sum_{n\leq i\leq m} |Z_i|.$$

On the right hand side every word $w \in \bigcup_{n\leq i\leq m} Z_i$ is counted with multiplicity t_w, where

$$t_w = |\{i : w \in Z_i\}|.$$

If $i \neq i'$, then $V(w,i) \neq V(w,i')$, because $|h(w,i) - h(w,i')| > 1$. Thus the leaf corresponding to w is counted on the left hand side at least t_w times. Therefore by using Claim 1 we get

$$\sum_{j\in J} d_j \geq \epsilon_3 n \binom{n+m}{n} = \epsilon_1 k \log k.$$

We now turn to the second condition of the Unbalance Lemma. Let $D_j = V(w,i)$ for some $w \in Z_i$, and let's suppose without loss of generality that the comparisons involving a_{h-1} preceede those involving a_h. Let a_{h+r} denote the leftmost right neighbour of a_h which has already been treated when the comparisons involving a_h begin at D_j. (If there is no such element we set $r = n - h + 1$). Let us further suppose that a_{h-1} and a_{h+r} are separated by s occurrences of b for some $s \geq 1$, which we denote by $b_e, \ldots, b_{e+s-1}$. Let us observe that $s/(r+s) = |z|_b/|z|$ for some initial segment z of xy, where z satisfies the conditions of Claim 2. Therefore we have

$$19/30 \leq s/(r+s) \leq 7/10.$$

The comparison at D_j is $a_h : b_{e+l-1}$ for some $1 \leq l \leq s$. Let $p(r,s,l) = Pr[a_h > b_{e+l-1}]$ denote the answer probability at D_j. Among the leaves of $V(w,i)$, the relative rankings of the sets $\{a_h, \ldots, a_{h+r-1}\}$ and $\{b_e, \ldots, b_{e+s-1}\}$ are all equally represented. For any word belonging to the leaves of D_j, the relative rank of a_h whithin the pooled set is at least l if and only if $a_h > b_{e+l-1}$. Thus we have

$$p(r,s,l) = \frac{s(s-1)\ldots(s-l+1)}{(r+s)(r+s-1)\ldots(r+s-l+1)}$$

We claim that for every l, the answer probability $p(r,s,l)$ falls outside the interval $(0.49, 0.51)$, thus we can choose for example $\epsilon_2 = 0.01$. We prove this in two cases according to the value of l. If $l = 1$ then $p(r,s,1) = s/(r+s) \geq 19/30$. If $l \geq 2$ then we have

$$p(r,s,l) \leq \left(\frac{s}{r+s}\right)^2 \leq 0.49,$$

and the result follows. □

4 Open Problems

We have shown how to significantly improve upon the tape merge algorithm when the ratio m/n is at least the golden ratio. An interesting open question is the value of the smallest ratio m/n where an improvement — probabilistic respectively deterministic — can be obtained. Although the lower bounds in [2,9] hold only for deterministic algorithms, we conjecture that not even a probabilistic algorithm can achieve improvements over tape merge for ratios less than the golden ratio.

It would be interesting to design other more complex probabilistic algorithms. Generalizing our average case lower bound for arbitrary merging algorithms also remains open.

Acknowledgements

Thanks are due to Charles Delorme for his most valuable assistance.

References

[1] C. Christen (1978) *Improving the bound on optimal merging*, Proceedings of 19th FOCS, pp. 259-266.

[2] C. Christen (1978) *On the optimality of the straight merging algorithm*, Publication 296, Dép. d'Info. et de Rech. Op., Université de Montréal.

[3] V. Chvatal (1984) *Probabilistic methods in graph theory*, Annals of Operations Research 1, pp. 171-182.

[4] F. K. Hwang and S. Lin (1971), *Optimal merging of 2 elements with n elements*, Acta Informatica 1, pp. 145-158.

[5] F. K. Hwang and S. Lin (1972), *A simple algorithm for merging two disjoint linearly ordered lists*, SIAM J. of Comput. 1, pp. 31-39.

[6] F. K. Hwang and D. N. Deutsch (1973) *A class of merging algorithms*, JACM, Vol 20, No. 1, pp. 148-159.

[7] D. E. Knuth (1973), The Art of Computer Programming, Volume 3: Sorting and Searching, Addison-Wesley.

[8] G. K. Manacher (1979), *Significant improvements to the Hwang-Ling merging algorithm*, JACM, Vol. 26, No. 3, pp. 434-440.

[9] P. K. Stockmeyer and F. F. Yao (1980) *On the optimality of linear merge*, SIAM J of Comput. 9, pp. 85-90.

[10] A. Yao (1977), *Probabilistic computations: Towards a unified mesure of complexity*, Proc. 18th FOCS, pp. 222-227.