On Probabilistic Networks for Selection, Merging, and Sorting

Tom Leighton [1] *Yuan Ma* [2] *Torsten Suel* [3]

Abstract

We study comparator networks for selection, merging, and sorting that output the correct result with high probability given a random input permutation. We prove tight bounds, up to constant factors, on the size and depth of probabilistic (n, k)-selection networks. In the case of $(n, n/2)$-selection, our result gives a somewhat surprising bound of $\Theta(n \log\log n)$ on the size of networks of success probability in $[\delta, 1 - 1/\mathrm{poly}(n)]$, where δ is an arbitrarily small positive constant, thus comparing favorably with the best previously known solutions, which have size $\Theta(n \log n)$. We also prove tight bounds, up to lower order terms, on the size and depth of probabilistic merging networks of success probability in $[\delta, 1-1/\mathrm{poly}(n)]$, where δ is an arbitrarily small positive constant. Finally, we describe two fairly simple probabilistic sorting networks of success probability at least $1 - 1/\mathrm{poly}(n)$ and nearly logarithmic depth.

[1] Department of Mathematics and Laboratory for Computer Science, MIT, Cambridge, MA 02139. Supported by ARPA Contracts N00014-91-J-1698 and N00014-92-J-1799. Email: `ftl@math.mit.edu`.

[2] Department of Computer Science, Stanford University, Stanford, CA 94305. Supported by an NSF Mathematical Sciences Postdoctoral Research Fellowship. Part of this work was done while the author was at MIT and supported by ARPA Contracts N00014-91-J-1698 and N00014-92-J-1799. Email: `yuan@cs.stanford.edu`.

[3] NEC Research Institute, 4 Independence Way, Princeton, NJ 08540. Part of this work was done while the author was at the University of Texas at Austin, and was supported by the Texas Advanced Research Program under Grant No. ARP-93-003658-461. Email: `torsten@research.nj.nec.com`.

1 Introduction

A *comparator network* is an acyclic circuit of comparators, each having two input registers and two output registers. One output register of a comparator is labeled as the *max-output*, and receives the larger of the two input values; the other is labeled as the *min-output*, and receives the smaller of the two input values.[1] The *size* of a network $\mathcal{N}$ is defined as the number of comparators in $\mathcal{N}$, and the *depth* of $\mathcal{N}$ is defined as the maximum length of a directed path in $\mathcal{N}$. An n-input comparator network is called a *sorting network* if it sorts all $n!$ permutations of $[n]$. (We use $[n]$ to denote the set $\{0, \ldots, n-1\}$.)

Sorting networks have been extensively studied in the past several decades, and they have important applications in the context of packet routing and circuit switching (e.g., see [14]). The readers are referred to [12, 19] for excellent surveys on this subject. Two sorting networks of particular importance are Batcher's networks [7] with size $(\frac{1}{4} + o(1)) \cdot n \log^2 n$ and depth $(\frac{1}{2} + o(1)) \cdot \log^2 n$, and the AKS network of Ajtai, Komlós, and Szemerédi [2, 3] with depth $O(\log n)$ and a large constant behind the O-notation.[2]

Comparator networks for the related problems of merging and selection have also been widely studied [5, 8, 9, 10, 12, 13, 18, 20, 21, 26, 27]. A *selection network* is a comparator network that classifies the input values into two groups such that all values in the first group are smaller than all those in the other group.[3] A *merging network* is a comparator network

[1] We often refer to a comparator network as a *network*, unless specified otherwise. An n-input comparator network can also be defined as a collection of n wires connected by comparators (e.g., see [12]). This model is equivalent to our model, which we have chosen to simplify the presentation of our proofs.

[2] We use "log" for the logarithm with base 2.

[3] A selection network is commonly defined as a network that classifies its input values into two groups, whereas the goal of a selection *algorithm* is to find an element of a given rank. When a network is required to work on *all* input permutations, it is well-known that the optimal size and depth of a "classifying" network differ only by a constant factor from those of a network for finding an element of a given rank [12]. Classifying networks

that merges two sorted lists.

More recently, comparator networks that correctly sort, select, or merge almost all input permutations have been studied. (See [8, 11, 15, 16, 22], and see [17, Chapter 5] for some simulation results due to Leighton, Ma, and Plaxton). Such networks are called *probabilistic networks*, because they output the correct result with high probability on a random input permutation.

The study of probabilistic sorting networks was initiated by Leighton and Plaxton [16], who designed a network of depth $(7.44 - o(1)) \log n$ that sorts the vast majority of all $n!$ permutations. This result was subsequently used by Plaxton [22] to prove the existence of a $2^{O(\sqrt{\log \log n})} \log n$ depth sorting network (in the strict sense) that can be efficiently implemented on the hypercube. Prior to this result, the AKS network, which has a complicated structure due to the use of expanders, was the only known sorting network of depth $o(\log^2 n)$.

In this paper, we are primarily concerned with the study of probabilistic networks for selection and merging, although we also describe some applications to sorting. The main results of the paper are as follows.

- We establish tight bounds, up to constant factors, on the size of probabilistic (n, k)-selection networks. Our size bound completely characterizes the optimal trade-off, up to a constant factor, between the size and the success probability of a probabilistic selection network. For the case of $(n, n/2)$-selection networks of success probability in $[\delta, 1-1/\text{poly}(n)]$, where δ is an arbitrarily small positive constant, our result gives a size bound of $\Theta(n \log \log n)$, while the best previous construction had a size of $O(n \log n)$ [8].

- We establish tight bounds, up to lower order terms, on the size and depth of probabilistic merging networks with success probability in $[\delta, 1 - 1/\text{poly}(n)]$, where δ is an arbitrarily small positive constant. For the case of two lists of equal length, the optimal size and depth of a probabilistic merging network improve by a factor of 2 over the best possible worst-case construction.

- An immediate consequence of the result on merging is a very simple variant of Batcher's classic odd-even sorting network, which has success probability at least $1 - 1/\text{poly}(n)$ but only half the size and depth of Batcher's original network. We also describe two simple probabilistic sorting networks of nearly logarithmic depth.

are also useful in the circuit switching networks, where it is desired to route the k smallest packets in one direction and the $n - k$ other packets in the other direction.

2 Notation

An n-input network $\mathcal{N}$ is called a *sorting network of success probability at least* $1 - \epsilon$ if there exists a fixed indexing of the output registers of $\mathcal{N}$ such that on at least $(1 - \epsilon)n!$ permutations, $\mathcal{N}$ outputs the ith smallest input value to the ith output register for all i. We say that $\mathcal{N}$ sorts a sequence of n numbers (not necessarily a permutation) if $\mathcal{N}$ outputs all of the numbers in the sequence in sorted order.

An n-input network $\mathcal{N}$ is called an (n, k)*-selection network of success probability at least* $1-\epsilon$ if the output registers of $\mathcal{N}$ can be partitioned into two sets L and S such that on at least $(1-\epsilon)n!$ permutations, $\mathcal{N}$ outputs the k smallest input values on registers in S, and the $n-k$ largest input values on registers in L. We also say that $\mathcal{N}$ selects a sequence of n numbers if $\mathcal{N}$ outputs the k smallest numbers of the sequence on registers in S and the $n - k$ largest numbers of the sequence on registers in L. We refer to a register in L as a *large output* and to a register in S as a *small output*. Throughout the paper, we assume $k \leq n/2$ (unless specified otherwise) since the other case is symmetric.

In a probabilistic (m_0, m_1)-merging network, the input permutation is drawn uniformly from the set of permutations of $[m_0 + m_1]$ where the values in the first m_0 input registers, and the values in the last m_1 input registers, respectively, each form a sorted list. Note that there are $\binom{m_0+m_1}{m_0}$ such permutations. An $(m_0 + m_1)$-input network $\mathcal{N}$ is called an (m_0, m_1)*-merging network of success probability at least* $1 - \epsilon$ if $\mathcal{N}$ merges at least $(1 - \epsilon)\binom{m_0+m_1}{m_0}$ of the permutations under consideration. Equivalently, we could assume that the input permutation is first drawn uniformly from all the $(m_0 + m_1)!$ permutations of $[m_0 + m_1]$, and then a fixed set of m_0 numbers and the set of the other m_1 numbers are independently sorted before entering $\mathcal{N}$. Throughout the paper, we assume $m_0 \leq m_1$ since the other case is symmetric.

We will often refer to such networks simply as *probabilistic* sorting, selection, or merging networks. In contrast, a network that works correctly on all possible input permutations (i.e., the ϵ in the above definitions is equal to zero) will be called a *worst-case* selection, merging, or sorting network.

Finally, we use $S(\mathcal{N})$ and $D(\mathcal{N})$ to denote the size and depth of a network $\mathcal{N}$, respectively.

3 Selection

Throughout this section, we consider (n, k)-selection networks with $k \leq \frac{n}{2}$; the other case is symmetric. The following theorems are the main results of the section, and they provide tight bounds, up to constant factors, on the size and depth of probabilistic (n, k)-selection networks.

Theorem 1 *For any $\epsilon < 1 - \delta$, where δ is an arbitrarily small positive constant, any (n, k)-selection network of success probability at least $1 - \epsilon$ must have $\Omega(n \log\log_{n/k}(n/(\epsilon + \binom{n}{k}^{-1})) + n)$ size and $\Omega(\log n)$ depth.*

Theorem 2 *There exists an explicit construction of an (n, k)-selection network of success probability at least $1 - \epsilon$ that has $O(n \log\log_{n/k}(n/(\epsilon + \binom{n}{k}^{-1})) + n)$ size and $O(\log n)$ depth.*

The following corollary can be viewed as a simplified version of the theorems.

Corollary 2.1 *For all k and all $\epsilon \in [1/\mathrm{poly}(n), 1-\delta]$, where δ is an arbitrarily small positive constant, the smallest (n, k)-selection network of success probability at least $1-\epsilon$ has $\Theta(n+n \log\log_{n/k} n)$ size and $\Theta(\log n)$ depth.*

In the bounds stated in Theorems 1 and 2, the $\binom{n}{k}^{-1}$ terms are included in the denominators to prevent the bounds from tending to infinity when $\epsilon \to 0$. Alternatively, we could have assumed $\epsilon \geq \frac{1}{2}\binom{n}{k}^{-1}$ in the theorems, since, by Lemma 3.1 and the fact that there are exactly $\binom{n}{k}$ 0-1 sequences of k 0s and $n - k$ 1s, an (n, k)-selection network with success probability strictly larger than $1 - \binom{n}{k}^{-1}$ must be a worst-case (n, k)-selection network. When $\epsilon = 0$, the bounds of Theorems 1 and 2 become the known $\Theta(n \log k)$ tight bound on the size of worst-case (n, k)-selection networks. (See Lemma 3.7 and the comments immediately before the lemma.)

Compared with the tight bound for worst-case (n, k)-selection networks, it is surprising that Corollary 2.1 states a tight bound of $\Theta(n \log\log n)$ on the size of $(n, n/2)$-selection networks of success probability in $[\delta, 1-1/\mathrm{poly}(n)]$, where δ is an arbitrarily small positive constant. The corollary also implies a surprising tight bound of $\Theta(n)$ on the size of (n, n^α)-selection networks of success probability in $[\delta, 1-1/\mathrm{poly}(n)]$, for any constant $\alpha < 1$.

We remark that our upper bound constructions use a worst-case (n, k)-selection network that is based on the $1.89\, n \log n$-size and $O(\log n)$-depth worst-case $(n, n/2)$-selection network of Jimbo and Maruoka [9] (see Lemma 3.7). Thus, our probabilistic network will have a moderate constant behind the O-notation in the size bound (the constant for the depth will be large), and it will use expanders or random graphs that are building blocks in the construction of the so-called ε-halvers [4]. In some special cases, we can avoid using worst-case selection networks (and their expander-based or random-graph-based structure) to achieve smaller constant factors in the size and depth bounds. For example, we can use the simple butterfly-based network of [16] to construct an $(n, n/2)$-selection network of success probability at least $1 - 1/\mathrm{poly}(n)$. However, if the probabilistic selection network is required to work with "extremely" high probability, then a simple construction becomes very difficult to find. This should not be surprising since all of the currently known $O(n \log n)$-size worst-case selection networks use expanders or random graphs and the optimal size and depth of worst-case selection networks are known only up to constant factors [1, 5, 9, 26]. Thus, in our upper bound constructions, we will not attempt to minimize the constant factors or to keep the network structure simple.

The rest of this section is devoted to the proofs of Theorems 1 and 2.

3.1 Lower Bound Proof

This subsection contains the proof of Theorem 1. We first introduce some notation and prove a few lemmas. An (n, k)-*sequence* is defined as a 0-1 sequence with k 0s and $n - k$ 1s. A *random* (n, k)-*sequence* is defined to be a sequence chosen uniformly at random from all of the $\binom{n}{k}$ (n, k)-sequences. Equivalently, it can be defined to be a sequence of n numbers that is randomly generated as follows: (1) choose a set of k positions uniformly at random from all of the n positions, and (2) set all of the numbers at these k positions to 0 and all of the other $n - k$ numbers to 1.

The *weight* of a register r in a network, denoted by $w(r)$, is defined inductively as follows: (1) $w(r) = 0$ if r is an input register; (2) if r is an output register of a comparator C with input registers x and y, then $w(r) = \min\{w(x), w(y)\}$ if r is the min-output of C, and $w(r) = 1+\max\{w(x), w(y)\}$ if r is the max-output of C. For any network $\mathcal{N}$, $W(\mathcal{N})$ is defined to be the sum of the weights of all output registers of $\mathcal{N}$.

For any register r in a network, we say that I, a set of input positions, is a *min term* of r if r will contain a 0 when a 0-1 sequence is input to the network such that all values input to I are 0s (independent of how many 0s and 1s are input to the other $n - |I|$ input registers). For example, the input register at position i has a unique min term $\{i\}$. The *size of a min term* I is defined to be $|I|$.

We next state a few useful lemmas. The proofs of these lemmas are not particularly difficult, but we will omit some of the proofs due to the limited space.

The first lemma is an analogue of the 0-1 principle [12], and it will enable us to study probabilistic selection networks by only considering their behavior on 0-1 input sequences.

Lemma 3.1 *A network is an (n, k)-selection network of success probability at least $1 - \epsilon$ if and only if it (n, k)-selects a random (n, k)-sequence with probability at least $1 - \epsilon$.*

Proof: For any 0-1 sequence $s = (s_1, s_2, \ldots, s_n)$, let $P(s)$ be the set of all permutations, $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, of $[n]$ such that $\pi_i < k$ if and only if $s_i = 0$. By the standard technique to prove the 0-1 principle, it is straightforward to check that a network (n, k)-selects s if and only if it (n, k)-selects all permutations in $P(s)$.

The set of all $n!$ permutations is now naturally partitioned into $\binom{n}{k}$ sets, each having the form of $P(s)$ and exactly $k!(n-k)!$ permutations. Now the correctness of the theorem follows from a straightforward counting argument. ■

Lemma 3.2 *Let X be an arbitrary set of input positions and s be a random (n, k)-sequence. Then (1) $\Pr[X$ contains $|X|$ 0s$] \le \left(\frac{k}{n}\right)^{|X|}$, and (2) $\Pr[X$ contains $|X|$ 0s$] \ge \left(\frac{k}{2n}\right)^{|X|}$ if $|X| \le \frac{k}{3}$.*

Proof: Clearly,

$$\begin{aligned} & \Pr[\text{X contains } |X| \text{ 0s}] \\ = \; & \frac{\binom{n-|X|}{k-|X|}}{\binom{n}{k}} = \frac{k(k-1)\cdots(k-|X|+1)}{n(n-1)\cdots(n-|X|+1)}. \end{aligned}$$

Now Property (1) follows from $\frac{k-i}{n-i} \le \frac{k}{n}$ for $i \ge 0$, and Property (2) follows from $\frac{k-i}{n-i} \ge \frac{k}{2n}$ for $0 \le i \le \frac{nk}{2n-k}$. ■

The next two lemmas are essentially due to Alekseyev (see [12]), and can be proved by induction on $S(\mathcal{N})$ and the depth of r, respectively.

Lemma 3.3 *For any network $\mathcal{N}$, $W(\mathcal{N}) = S(\mathcal{N})$.*

Lemma 3.4 *Any register r has a min term of size at most $2^{w(r)}$.*

The intuition of the following lemma is simple: the number of 1s is fixed in a random (n, k)-sequence, and therefore the event that some sets of positions contain 1s makes all other positions less likely to contain 1s.

Lemma 3.5 *Let $X_1, X_2, \ldots, X_m$ be m sets of input positions such that $X_i \cap X_j = \emptyset$ for $1 \le i < j \le m$. Then on a random (n, k)-sequence,*

$$\begin{aligned} & \Pr[X_i \textit{ contains at least one 1 for all } i \le m] \\ \le \; & \prod_{1 \le i \le m} \Pr[X_i \textit{ contains at least one 1}]. \end{aligned}$$

Proof of Theorem 1: Consider an arbitrary (n, k)-selection network $\mathcal{N}$ that has success probability at least $1 - \epsilon$. We need to prove that both $D(\mathcal{N})$ and $S(\mathcal{N})$ must be sufficiently large.

To prove $D(\mathcal{N}) = \Omega(\log n)$, we make use of the following observation: If the ith and the jth input registers cannot reach the same comparator in $\mathcal{N}$ (thus, the input values to these two positions will never be compared in $\mathcal{N}$), then $\mathcal{N}$ will fail to (n, k)-select either an (n, k)-sequence $x = (x_1, x_2, \ldots, x_n)$ or the sequence generated from x by swapping x_i and x_j where $x_i \ne x_j$. The details will be included in the full paper. We remark that the $\Omega(\log n)$ depth lower bound would be trivial if a selection network were defined to be a network for finding an element of a given rank,

In the remainder of the proof, we will prove that $S(\mathcal{N})$ must be sufficiently large. Let L be the set of $n-k$ large outputs of $\mathcal{N}$. By the remarks immediately below Corollary 2.1, we will assume without loss of generality

$$\epsilon \ge \frac{1}{2}\binom{n}{k}^{-1}. \qquad (1)$$

By the definition of $W(\mathcal{N})$ and Lemma 3.3, $\mathcal{N}$ contains an output register $r_0 \in L$ such that $w(r) \le \frac{W(\mathcal{N})}{|L|} = \frac{S(\mathcal{N})}{|L|}$. Hence, by Lemma 3.4, register r_0 has a min term A_0 of size

$$|A_0| \le 2^{\frac{S(\mathcal{N})}{|L|}}. \qquad (2)$$

We next use the preceding inequality to prove

$$S(\mathcal{N}) = \Omega\left(n \log\log_{\frac{n}{k}} \frac{1}{\epsilon}\right). \qquad (3)$$

Clearly, Inequality 2 implies

$$S(\mathcal{N}) \geq |L| \log |A_0| \geq \frac{n}{2} \log |A_0|.$$

Hence, it suffices to show

$$|A_0| = \Omega\left(\log_{\frac{n}{k}} \frac{1}{\epsilon}\right). \qquad (4)$$

If $|A_0| \geq \frac{k}{3}$, then using $\binom{n}{k} \leq (\frac{en}{k})^k$ and Inequality 1, we obtain $|A_0| = \Omega\left(\log_{\frac{n}{k}} \left(\frac{en}{k}\right)^k\right) = \Omega\left(\log_{\frac{n}{k}} \binom{n}{k}\right) = \Omega\left(\log_{\frac{n}{k}} \frac{1}{\epsilon}\right)$, as desired. We next assume $|A_0| \leq \frac{k}{3}$. By the choice of r_0 and A_0, $\mathcal{N}$ will fail to (n,k)-select a random (n,k)-sequence s if all inputs to A_0 on s are 0s. By Lemma 3.2 and the assumption $|A_0| \leq \frac{k}{3}$, this happens with probability at least $(\frac{k}{2n})^{|A_0|}$. On the other hand, by Lemma 3.1, $\mathcal{N}$ fails to (n,k)-select s with probability at most ϵ. Hence, we have $(\frac{k}{2n})^{|A_0|} \leq \epsilon$, which implies $|A_0| \geq \log_{\frac{2n}{k}} \frac{1}{\epsilon} = \Omega(\log_{\frac{n}{k}} \frac{1}{\epsilon})$, as desired.

We have thus established Inequality 3. To complete the lower bound proof for $S(\mathcal{N})$, we still need to show

$$S(\mathcal{N}) = \Omega(n), \quad \text{and} \qquad (5)$$

$$S(\mathcal{N}) = \Omega\left(n \log\log_{\frac{n}{k}} n\right). \qquad (6)$$

It is easy to see that Equation 6 is stronger than Equation 5 if and only if $k \geq \sqrt{n}$. Hence, it suffices to prove Equation 5 for $k < \sqrt{n}$ and Equation 6 for $k \geq \sqrt{n}$.

The proof of Equation 5 for $k < \sqrt{n}$ is quite straightforward. Assume for contradiction that $S(\mathcal{N}) \leq \frac{\delta}{4}n$. Then, it is easy to see that at least $n - k - \frac{\delta n}{2}$ input registers of $\mathcal{N}$ are used directly as large output registers without passing any comparators. However, on a random (n,k)-sequence s, the probability that at least one of these $n - k - \frac{\delta n}{2}$ registers contains a 0 is at least

$$1 - \frac{\binom{k+\delta n/2}{k}}{\binom{n}{k}} \geq 1 - \frac{k + \delta n/2}{n} \geq 1 - \delta > \epsilon,$$

where the second inequality follows from $k/n < 1/\sqrt{n} < \delta/2$ for sufficiently large n. This contradicts the fact that $\mathcal{N}$ fails to (n,k)-select s with probability at most ϵ.

In what follows, we prove Equation 6 under the assumption $k \geq \sqrt{n}$. We will construct a sequence of subnetworks, $\mathcal{N}_0, \mathcal{N}_1, \ldots, \mathcal{N}_{i_0}$, of $\mathcal{N}$, together with three related sequences $L_0, L_1, \ldots, L_{i_0}$, $r_0, r_1, \ldots, r_{i_0}$, and $A_0, A_1, \ldots, A_{i_0}$ where $L_i = L \cap \mathcal{N}_i$, $r_i \in L_i$, and A_i is a min term of r_i.

First, we set $\mathcal{N}_0 = \mathcal{N}$ and $L_0 = L$. Let r_0 and A_0 be defined as immediately before Inequality 2.

We next construct a network $\mathcal{N}_1$ from $\mathcal{N}_0$ as follows. First set each input in A_0 to 1. Then propagate these $|A_0|$ 1s through $\mathcal{N}_0$ by letting a 1 always go to the max output of a comparator if the comparator's other input is not specified. Remove all of the registers containing a 1 and all of the comparators comparing 1 and 1. Replace each of the comparators that contain exactly one 1 input by directly connecting its non-1 input to its non-1 output. Let $\mathcal{N}_1$ be the resulting network.

$\mathcal{N}_1$ can be naturally viewed as a subnetwork of $\mathcal{N}_0$, and each register of $\mathcal{N}_1$ naturally corresponds to a register of $\mathcal{N}_0$. Thus, we can define $L_1 = L_0 \cap \mathcal{N}_1$. If $L_1 \neq \emptyset$, then $\mathcal{N}_1$ contains an output register $r_1 \in L_1$ such that $w_1(r_1) \leq \frac{W(\mathcal{N}_1)}{|L_1|} = \frac{S(\mathcal{N}_1)}{|L_1|}$, where $w_1(r_1)$ denotes the weight of r_1 in $\mathcal{N}_1$ (as opposed to the weight in $\mathcal{N}$) and the equality follows from Lemma 3.3. Hence, by Lemma 3.4, we can choose a min term A_1 of r_1 in $\mathcal{N}_1$ such that $|A_1| \leq 2^{\frac{S(\mathcal{N}_1)}{|L_1|}}$. It is easy to check that A_1 is also a min term of r_1 in $\mathcal{N}$ and $A_1 \cap A_0 = \emptyset$.

In general, we construct $\mathcal{N}_i$ from $\mathcal{N}_{i-1}$ as follows: (1) set each input in A_{i-1} to 1 and propagate these 1s through $\mathcal{N}_{i-1}$; (2) remove all of the registers containing a 1 and all of the comparators comparing 1 and 1; (3) replace each of the comparators that contain exactly one 1 input by directly connecting its non-1 input to its non-1 output.

Let $L_i = L_{i-1} \cap \mathcal{N}_i$ (which is equal to $L \cap \mathcal{N}_i$). If $L_i \neq \emptyset$, then by Lemma 3.3, we can choose an output register $r_i \in L_i$ such that $w_i(r_i) \leq \frac{W(\mathcal{N}_i)}{|L_i|} = \frac{S(\mathcal{N}_i)}{|L_i|}$ where $w_i(r_i)$ denotes the weight of r_i in $\mathcal{N}_i$, as opposed to the weight in $\mathcal{N}$. Hence, by Lemma 3.4, we can choose a min term A_i of r_i in $\mathcal{N}_i$ such that

$$|A_i| \leq 2^{\frac{S(\mathcal{N}_i)}{|L_i|}}. \qquad (7)$$

Clearly,

$$A_i \cap A_j = \emptyset \quad \text{for } 0 \leq j < i, \quad \text{and} \qquad (8)$$

$$|L_i| \geq |L| - \sum_{0 \leq j < i} |A_j|. \qquad (9)$$

Most importantly, however, it is easy to check that A_i is also a min term of $r_i \in L$ in $\mathcal{N}$. Hence, if a random (n,k)-sequence s makes all of the input registers in A_i contain 0s, then $\mathcal{N}$ will not (n,k)-select s.

We can continue this procedure as long as $|L_i| \ge 1$. In what follows, we choose i_0 to be the largest i such that $|L_i| \ge \frac{n}{4}$.

If $|A_i| \ge \frac{\sqrt{n}}{4}$ for some $i \le i_0$, then Equation 6 immediately follows from Inequality 7. Moreover, if $|A_i| \ge \frac{k}{3}$ for some $i \le i_0$, then Equation 6 follows from Inequality 7 together with the inequality $\frac{k}{3} \ge \log_{\frac{n}{k}} n$ for $k \ge \sqrt{n}$.

Thus, in what follows, we will assume $|A_i| \le \min\{\frac{k}{3}, \frac{\sqrt{n}}{4}\}$ for $i \le i_0$. By Inequality 9 and $|L| = n - k \ge n/2$, this implies $i_0 \ge \sqrt{n}$. By the choice of r_i and A_i, $\mathcal{N}$ will fail to (n,k)-select a random (n,k)-sequence if there exists $i \le \sqrt{n}$ such that all of the inputs to A_i are 0s. Hence,

$$\begin{aligned}
& 1-\epsilon \\
\le\ & \Pr[\,\mathcal{N} \text{ selects } s\,] \\
\le\ & \Pr[\,\text{at least one input to } A_i \text{ is } 1\ \forall\, i \le \sqrt{n}\ ] \\
\le\ & \prod_{0 \le i \le \sqrt{n}} \Pr[\,A_i \text{ contains at least one } 1\,] \\
& \text{(by Lemma 3.5)} \\
\le\ & \prod_{0 \le i \le \sqrt{n}} \left(1 - \left(\frac{k}{2n}\right)^{|A_i|}\right) \\
& \text{(by Lemma 3.2)} \\
\le\ & \prod_{0 \le i \le \sqrt{n}} \exp\left(-\left(\frac{k}{2n}\right)^{|A_i|}\right) \\
& \text{(by } 1 - x \le e^{-x}\text{)} \\
\le\ & \exp\left(-\sqrt{n}\cdot\left(\frac{k}{2n}\right)^{2^{\frac{4S(\mathcal{N})}{n}}}\right) \\
& \text{(by Inequality 7).}
\end{aligned}$$

Thus, $\sqrt{n}\cdot(\frac{k}{2n})^{2^{\frac{4S(\mathcal{N})}{n}}} \le \ln\frac{1}{1-\epsilon} \le \ln\frac{1}{\delta}$, which implies Equation 6. ■

3.2 Upper Bound Proof

This subsection contains the proof of Theorem 2. We first prove two useful lemmas.

Lemma 3.6 *For an arbitrary set of m input positions X, a random (n,k)-sequence s, and $t > 0$, the probability that X contains at least $km/n + t$ 0s or at least $(n-k)m/n + t$ 1s is at most $O(\sqrt{n})\exp(-2t^2/m)$.*

Proof: Let $\mathrm{P}_1[A]$ and $\mathrm{P}_2[A]$ denote the probability that an event A occurs when (1) each of the inputs is independently set to 0 with probability k/n and to 1 with probability $1-k/n$ and (2) the input sequence is a random (n,k)-sequence, respectively. Let E denote the event that X contains at least $km/n + t$ 0s or at least $(n-k)m/n + t$ 1s. Then, we have

$$\begin{aligned}
& \mathrm{P}_1[E] \\
\ge\ & \mathrm{P}_2[E]\cdot \mathrm{P}_1[\text{the input is an } (n,k)\text{-sequence}] \\
=\ & \mathrm{P}_2[E]\binom{n}{k}\left(\frac{k}{n}\right)^k\left(1-\frac{k}{n}\right)^{n-k} \\
=\ & \mathrm{P}_2[E]\ \Omega\left(\frac{1}{\sqrt{n}}\right),
\end{aligned}$$

where the last equality follows from Stirling's approximation and simple calculations. Now the correctness of the lemma follows from the Chernoff bound in [6, Corollary A.7 on page 236]. ■

In the next lemma, we describe a worst-case (n,k)-selection network of $O(n\log k)$ size and $O(\log n)$ depth. The depth of such a network is apparently optimal, up to a constant factor. Compared with the $\Omega(n\log k)$ size lower bound established by Alekseyev [5, 12], the size of the network is also optimal, up to a constant factor. Given the simple nature of our network, it is likely that this lemma has been discovered before. However, we have not been able to find any pointer in the literature, and thus we will give our own proof. Endre Szemerédi pointed out that the lemma can be proved by using the new ε-halver construction of [4]. But such a proof only shows the existence of a desired network, whereas our proof gives an explicit construction.

Lemma 3.7 *There exists an explicit construction of a worst-case (n,k)-selection network of $O(n\log k)$ size and $O(\log n)$ depth.*

Proof: Imagine that all of the n input numbers are arranged in a binary tree of $\lceil\log\frac{n}{k}\rceil$ height so that each of the nodes in the tree has exactly k numbers, except that the rightmost leaf contains m numbers with $m \le k$. We label all of the nodes at an arbitrary level of the tree from left to right by $1, 2, 3, \ldots, i$, for some $i \le n$.

The network proceeds in $\lceil\log\frac{n}{k}\rceil - 1$ rounds, each of which consists of 2 phases. In the first phase of the ith round ($1 \le i \le \lceil\log\frac{n}{k}\rceil - 1$), we apply a worst-case $(2k,k)$-selection network of $O(k\log k)$ size and $O(\log k)$ depth (e.g., the network described in [9]) to the numbers contained in each node with an odd label at the ith level from the bottom and its parent. In the second phase of the ith round, we apply a worst-case $(2k, k)$-selection network of $O(k \log k)$ size and $O(\log k)$ depth to the numbers contained in each node with an even label at the ith level from the bottom and its parent. (The network to the rightmost leaf and its parent should be an $(m+k, k)$-selection network, where $m \le k$ is the number of input values contained in that leaf.) When we apply a selection network to a node and its parent, we assume that the small numbers are moved to the parent.

It is straightforward to show that the network thus constructed has the desired depth and size and always moves the smallest k numbers to the root of the tree. ■

Proof of Theorem 2: By the remarks immediately below Corollary 2.1, we will assume without loss of generality $\epsilon \ge \frac{1}{2}\binom{n}{k}^{-1}$, since otherwise we can simply use a worst-case (n, k)-selection network of Lemma 3.7. Under this assumption, we need to construct a network $\mathcal{N}$ with depth $O(\log n)$ and size

$$O\left(n + n \log \log_{\frac{n}{k}} \frac{n}{\epsilon}\right). \qquad (10)$$

We construct $\mathcal{N}$ in two cases: $k \ge \frac{n}{\log^2 n}$ and $k \le \frac{n}{\log^2 n}$.

Case 1: $k \ge \frac{n}{\log^2 n}$. Let

$$m = \Theta\left(\log^4 \frac{n}{\epsilon}\right), \text{ and } t = \Theta\left(\log^3 \frac{n}{\epsilon}\right). \qquad (11)$$

If $m \ge n$, then $n/\epsilon = 2^{\Omega(n^{1/4})}$. Thus, Expression 10 becomes $O(n \log n)$, and we can choose $\mathcal{N}$ as a worst-case (n, k)-selection network of Lemma 3.7. In what follows, we will assume $m \le n$. Our network $\mathcal{N}$ is constructed as follows. Group all of the input registers into sets, $A_1, A_2, \ldots, A_{\lfloor n/m \rfloor}$, of size m each. (The last set may have m' registers with $m \le m' < 2m$, but we will not distinguish m' from m since the key fact to be used about m is in equation 11, which also holds for m'.) First, we apply a worst-case $(m, km/n + t)$-selection network $\mathcal{S}_i$ of Lemma 3.7 to each A_i, and let each of the $(n-k)m/n - t$ large outputs of $\mathcal{S}_i$ be a large output of $\mathcal{N}$. Next, we apply a worst-case $(km/n + t, \max\{km/n - t, 0\})$-selection network $\mathcal{S}'_i$ of Lemma 3.7 to the $km/n + t$ small outputs of each $\mathcal{S}_i$, and let each of the $\max\{km/n - t, 0\}$ small output registers of $\mathcal{S}'_i$ be a small output of $\mathcal{N}$. Let ℓ denote the number of the remaining outputs from all $\mathcal{S}'_i$ ($i \le \lfloor n/m \rfloor$). Then, $\ell = \lfloor n/m \rfloor (km/n + t - \max\{km/n - t, 0\}) \le 2tn/m$. Apply a worst-case $(\ell, \ell - t\lfloor n/m \rfloor)$-selection network $\mathcal{S}$ of Lemma 3.7 to all of these ℓ registers. Let each of $\ell - t\lfloor n/m \rfloor$ small outputs of $\mathcal{S}$ be a small output of $\mathcal{N}$, and let each of the $t\lfloor n/m \rfloor$ large outputs of $\mathcal{S}$ be a large output of $\mathcal{N}$.

It is straightforward to see that network $\mathcal{N}$ has an overall depth of $O(\log n)$ and an overall size of

$$\begin{aligned} S(\mathcal{N}) &= 2\left(\frac{n}{m}\right) O(m \log m) + O\left(\frac{2tn}{m} \log \frac{2tn}{m}\right) \\ &= O\left(n \log \log \frac{n}{\epsilon}\right) && \text{by Equation 11} \\ &= O\left(n \log \log_{\frac{n}{k}} \frac{n}{\epsilon}\right) && \text{for } k \ge \tfrac{n}{\log^2 n}. \end{aligned}$$

It remains to show that $\mathcal{N}$ is an (n, k)-selection network of probability at least $1 - \epsilon$. By Lemma 3.1, we only need to show that $\mathcal{N}$ selects a random (n, k)-sequence s with probability at least $1 - \epsilon$. Note that $\mathcal{N}$ consists of three rounds of worst-case selection networks. Hence, $\mathcal{N}$ (n, k)-selects a random (n, k)-sequence s if after the applications of $\mathcal{S}_i$ and $\mathcal{S}'_i$ for all i, (1) all of the $n - k$ 1s are either input to $\mathcal{S}$ or output by one of the $(n-k)m/n - t$ large outputs of some $\mathcal{S}_i$, and (2) all of the k 0s are either input to $\mathcal{S}$ or output by one of the $\max\{km/n - t, 0\}$ small outputs of some $\mathcal{S}'_i$. This happens when for all i A_i contains at most $km/n + t$ 0s and at most $(n-k)m/n + t$ 1s. In other words, $\mathcal{N}$ fails to (n, k)-select s only if there exists i such that A_i contains more than $km/n + t$ 0s or more than $(n-k)m/n + t$ 1s. By Lemma 3.6, the probability that there exists $i \le n/m$ such that A_i contains more than $km/n + t$ 0s or more than $(n-k)m/n + t$ 1s is at most

$$\left(\frac{n}{m}\right) O(\sqrt{n}) \exp\left(-\frac{2t^2}{m}\right) \le \epsilon,$$

for sufficiently large n. This completes the analysis of network $\mathcal{N}$ and proves the theorem for Case 1.

Case 2: $k \le \frac{n}{\log^2 n}$. Let

$$l = \Theta\left(2 + \log_{\frac{n}{k}} \frac{n}{\epsilon}\right), \text{ and } m = l\,(3 + \log k). \qquad (12)$$

Without loss of generality, we will assume $m \le n$ since otherwise Expression 10 becomes $O(n \log n)$ and we can choose $\mathcal{N}$ as a worst-case (n, k)-selection network of Lemma 3.7. Our network $\mathcal{N}$ is constructed as follows. Group all of the input registers into groups, $A_1, A_2, \ldots, A_{\lfloor n/m \rfloor}$, of size m each. (The last set may have $m' \ge m$ registers, but we will not distinguish m' from m, as in Case 1.) Apply an (m, l)-selection network $\mathcal{S}_i$ of Lemma 3.7 to each A_i, and let each of the $m - l$ large outputs of $\mathcal{S}_i$ be a large output of $\mathcal{N}$. Apply an $(\lfloor n/m \rfloor l, k)$-selection network $\mathcal{S}$ of Lemma 3.7 to all of the $\lfloor n/m \rfloor\, l$ small outputs of all $\mathcal{S}_i$ ($i \le \lfloor n/m \rfloor$). Let each of the k small outputs of $\mathcal{S}$ be a small output of $\mathcal{N}$, and let each of the large outputs of $\mathcal{S}$ be a large output of $\mathcal{N}$.

The depth of $\mathcal{N}$ is apparently $O(\log n)$, and the size of $\mathcal{N}$ is $O(n \log l + \lceil n/m \rceil\, l \log k)$, which is equal to Expression 10 by Equation 12. Hence, by Lemma 3.1, we only need to show that $\mathcal{N}$ (n,k)-selects a random (n,k)-sequence s with probability at least $1-\epsilon$.

Note that network $\mathcal{N}$ consists of two rounds of worst-case selection networks. If all of the k 0s of s are input to $\mathcal{S}$, then $\mathcal{N}$ will (n,k)-select s. Moreover, if for all i, A_i contains at most l 0-inputs, then all of the 0s will be output from some $\mathcal{S}_i$ to $\mathcal{S}$. Hence, $\mathcal{N}$ will fail to (n,k)-select s only if there exists i such that A_i contains more than l 0s on input sequence s. By Lemma 3.2, the probability that there exists $i \le n/m$ such that A_i contains more than l 0s is at most

$$\left(\frac{n}{m}\right)\binom{m}{l}\left(\frac{k}{n}\right)^l.$$

Hence, we only need to prove

$$\left(\frac{em}{l}\right)^l\left(\frac{k}{n}\right)^l \le \frac{\epsilon}{n}.$$

By Equation 12, it suffices to prove

$$\left(\frac{em}{l}\right)^l\left(\frac{k}{n}\right)^{\frac{2l}{3}} \le 1,$$

which can be established by proving

$$\left(\frac{em}{l}\right)^3\left(\frac{k}{n}\right)^2 \le 1.$$

By Equation 12, it suffices to prove

$$(e(\log k + 3))^3\left(\frac{k}{n}\right)^2 \le 1.$$

When $k = \frac{n}{\log^2 n}$, the preceding inequality trivially holds for sufficiently large n. Since the left-hand side of the inequality is an increasing function of k, the inequality holds for all $k \le \frac{n}{\log^2 n}$. This completes the analysis of network $\mathcal{N}$ and proves the theorem for Case 2. ■

4 Merging and Sorting

In this section, we establish tight bounds on the size and depth of probabilistic (m_0, m_1)-merging networks, up to lower order terms, for a large range of success probabilities and sublist lengths, m_0 and m_1. We also give a simplified proof of the tight lower bound for worst-case merging networks established in [18], and describe two simple constructions of small-depth probabilistic sorting networks based on ideas related to merging.

Throughout the section, we assume $m_0 \le m_1$, since the other case is symmetric. We continue to use n to denote the total number of inputs to a merging network. Thus, $n = m_0 + m_1$.

4.1 Lower Bounds

In this subsection, we first prove our lower bounds for probabilistic merging networks, and then extend our method to worst-case merging networks.

Our proof technique is similar to the approach proposed by Pippenger and Valiant [21] (and also used in [13, 18, 23]), and extends to a very general class of comparison-based merging algorithms on fixed-connection networks. For simplicity, we will not attempt to obtain the best possible lower order terms in our bounds.

Consider a network of $\nu \ge n$ processors connected by a system of directed communication links. To distinguish it from a comparator network, we will refer to such a network as a *processor network*. Let m_0 and m_1 be integers with $m_0 \le m_1$ and $m_0 + m_1 = n$, and let $M_0 = \{x_0, \ldots, x_{m_0-1}\}$, $M_1 = \{y_0, \ldots, x_{m_1-1}\}$, and $N = \{z_0, \ldots, z_{n-1}\}$ be three arbitrary, fixed subsets of the set of processors. Then the *(m_0, m_1)-merging problem on the processor network* is defined as follows. The input consists of two sorted lists, one initially located in processors x_0 to x_{m_0-1}, and the other one initially located in processors y_0 to y_{m_1-1}, each in ascending order. The output consists of a single sorted list of length n contained in processors z_0 to z_{n-1}. Note that a processor is allowed to hold more than one element during the execution of the algorithm.

The next lemma gives a lower bound on the number of *data transports* needed to move the elements to their correct locations in the final sorted order, where each *data transport* consists of the transmission of a single element across some communication link.

Lemma 4.1 *For $m_0 = \omega(1)$, if an algorithm correctly merges a constant fraction of all possible input permutations for the (m_0, m_1)-merging problem on a processor network of out-degree at most $d \ge 2$, then on some input permutation that the algorithm correctly merges, the algorithm must make at least $(\frac{1}{2} - o(1)) \cdot n \log_d m_0$*

data transports.

Proof: Assume that the input permutations are drawn uniformly at random from the set of permutations of $[n]$ with the property that the elements in processors x_0 to x_{m_0-1} and y_0 to y_{m_1-1}, respectively, each form a sorted list. Let $p(x, z_j)$ denote the probability that the element initially located in processor x has to be moved to processor z_j (and, thus, has a global rank of j).

Let i be any integer such that $\frac{m_0}{\alpha} \le i < m_0 - \frac{m_0}{\alpha}$, where $\alpha = \omega(1)$ is to be chosen later. Then for all j with $j < i$ or $j > n - m_0 + i$, we have $p(x_i, z_j) = 0$. For all other j, we have

$$p(x_i, z_j) = \frac{\binom{j}{i}\binom{n-j-1}{m_0-i-1}}{\binom{n}{m_0}} = \frac{m_0}{n} \cdot \frac{\binom{j}{i}\binom{n-j-1}{m_0-i-1}}{\binom{n-1}{m_0-1}}.$$

By Stirling's approximation and some nontrivial calculations, it can be shown that

$$\frac{\binom{j}{i}\binom{n-1-j}{m_0-1-i}}{\binom{n-1}{m_0-1}} \le \Theta(1)\sqrt{\frac{(n-1)\,(m_0-1)}{m_1\, i\,(m_0-1-i)}}.$$

(Roughly, for fixed m_0, n, and i, the left-hand side of the inequality is maximized when j/i is approximately equal to $(n-1-j)/(m_0-1-i)$; a formal proof will be included in the full paper.) Hence,

$$p(x_i, z_j) \le \Theta(1)\sqrt{\frac{m_0\alpha}{n\, m_1}},$$

since $\frac{m_0}{\alpha} \le i < m_0 - \frac{m_0}{\alpha}$.

Let Δ be the set of input permutations that the algorithm merges correctly, and let $\delta = |\Delta|/\binom{n}{m_0}$, which is a constant by our assumption on the algorithm. If $p'(x_i, z_j)$ denotes the probability that the element initially located in x_i has a global rank of j, assuming that the input permutation is drawn uniformly at random from Δ, then clearly $p'(x_i, z_j) \le p(x_i, z_j)/\delta$.

Now let $S(x, c)$ be the set of processors that are reachable from processor x in less than c steps. Since the out-degree of the processor network is upper-bounded by $d \ge 2$, $S(x, c)$ contains at most $\sum_{0\le i<c} d^i < d^c$ processors. Hence, for $c = \log_d(\sqrt{\frac{n\,m_1}{m_0\alpha^3}})$, we have

$$\sum_{z_j \in S(x_i,c)} p'(x_i, z_j) \;\le\; \frac{d^c}{\delta} \cdot \sqrt{\frac{m_0\alpha}{n\,m_1}} \;=\; \frac{1}{\delta\alpha}.$$

Thus, by Markov's inequality, the expected number of data transports required to move an element from a processor x_i with $\frac{m_0}{\alpha} \le i < m_0 - \frac{m_0}{\alpha}$ to its destination is lower-bounded by

$$\begin{aligned} &\left(1 - \frac{1}{\delta\alpha}\right) \cdot \log_d\left(\sqrt{\frac{n\,m_1}{m_0\alpha^3}}\right) \\ = \;& (1 - o(1)) \cdot \log_d\left(\sqrt{\frac{n\,m_1}{m_0\alpha^3}}\right), \end{aligned} \tag{13}$$

where we have used $\alpha = \omega(1)$. Here and in the rest of the proof, all expectations are taken under the assumption that the input permutation is drawn uniformly at random from Δ.

Using an entirely symmetric argument, we know that the expected number of data transports required to move an element from a processor y_i with $\frac{m_1}{\alpha} \le i < m_1 - \frac{m_1}{\alpha}$ to its destination is lower-bounded by

$$(1 - o(1)) \cdot \log_d\left(\sqrt{\frac{n\,m_0}{m_1\alpha^3}}\right).$$

Now let $\alpha = \log m_0$. (This does not contradict the requirement $\alpha = \omega(1)$ since $m_0 = \omega(1)$.) By linearity of expectation, the expected number of data transports required to move all elements to their destinations is at least

$$\begin{aligned} & (1 - o(1)) \cdot m_0 \log_d\left(\sqrt{\frac{n\,m_1}{m_0\alpha^3}}\right) \\ & + (1 - o(1)) \cdot m_1 \log_d\left(\sqrt{\frac{n\,m_0}{m_1\alpha^3}}\right) \\ \ge\; & (1 - o(1)) \cdot m_0 \log_d\left(\sqrt{\frac{n\,m_0}{m_1\alpha^3}}\right) \\ & + (1 - o(1)) \cdot m_1 \log_d\left(\sqrt{\frac{n\,m_0}{m_1\alpha^3}}\right) \\ \ge\; & (1/2 - o(1)) \cdot n \log_d m_0, \end{aligned}$$

where the second inequality follows from $\log\alpha = o(\log m_0)$. Thus, by averaging, there exists an input permutation in Δ that requires at least $(1/2 - o(1)) \cdot n \log_d m_0$ data transports. ■

Theorem 3 *For $m_0 = \omega(1)$, any (m_0, m_1)-merging network with at least a constant success probability has size at least $(\frac{1}{4} - o(1)) \cdot n \log m_0$ and depth at least $(1 - o(1)) \cdot \log(\frac{n}{\sqrt{m_0}})$.*

Proof: Consider a comparator network as a processor network where the comparators and the input and output nodes of the comparator network are considered to be processors, and the registers of the comparator network are considered to be communication links. In such a processor network, each input node has out-degree 1, and each comparator has out-degree 2. Thus, the claimed size lower bound follows directly from Lemma 4.1 (note that each comparator performs exactly two data transports). Finally, the claimed depth lower bound follows from the lower bound on the number of data transports required to move a single element to its destination, as given by Equation 13. ■

We next use a similar argument to show a tight size lower bound (up to a lower order term) of $(\frac{1}{2} - o(1)) \cdot n \log m_0$ for worst-case (m_0, m_1)-merging networks with $m_0 = \omega(1)$. While such a bound was already established by Miltersen, Paterson, and Tarui [18], we believe that the new proof may still be of interest to the reader, as it is significantly simpler than the one in [18]. (However, our technique does not achieve the same lower order terms.) The lower bound follows directly from the next lemma.

Lemma 4.2 *For $m_0 = \omega(1)$, any algorithm for the (m_0, m_1)-merging problem on a processor network of out-degree $d \geq 2$ requires at least $(1 - o(1)) \cdot n \log_d m_0$ data transports in the worst case.*

Proof: Let π_k be the input permutation that assigns the smallest k values to $x_0, \ldots, x_{k-1}$, the next $m_1 - k$ values to $y_0, \ldots, y_{m_1-k-1}$, the next $m_0 - k$ values to $x_k, \ldots, x_{m_0-1}$, and finally the largest k values to $y_{m_1-k}, \ldots, y_{m_1-1}$. We show a lower bound on the expected number of data transports when the input permutation is drawn uniformly at random from the set $\Delta = \{\pi_k \mid k < \frac{m_0}{\log m_0}\}$. The lemma then follows from the observation that any lower bound on the expected number of data transports implies an identical lower bound for the worst case.

Let i be any integer such that $\frac{m_0}{\log m_0} \leq i < m_0$. Note that x_i receives a different input value under every permutation in Δ; this implies that $p(x_i, z_j) \leq \frac{1}{|\Delta|} = \frac{\log m_0}{m_0}$ holds for all j. Recall that $|S(x, c)| < d^c$, where $S(x, c)$ is the set of processors that are reachable from x in less than c steps, and d is an upper bound on the out-degree of the processor network. For $c = \log_d(\frac{m_0}{\log^2 m_0})$, we have

$$\sum_{z_j \in S(x_i, c)} p(x_i, z_j) \;\leq\; d^c \cdot \frac{\log m_0}{m_0} \;=\; \frac{1}{\log m_0}.$$

Thus, by Markov's inequality, the expected number of data transports to move an element from a processor x_i with $\frac{m_0}{\log m_0} \leq i < m_0$ to its destination is lower-bounded by $(1 - \frac{1}{\log m_0}) \cdot \log_d(\frac{m_0}{\log^2 m_0})$; the same bound also holds for all elements initially in a processor y_i with $0 \leq i < m_1 - \frac{m_0}{\log m_0}$. By linearity of expectation, the expected number of data transports required to move all elements to their destinations is at least

$$\left(n - \frac{2m_0}{\log m_0}\right) \cdot \left(1 - \frac{1}{\log m_0}\right) \cdot \log_d\left(\frac{m_0}{\log^2 m_0}\right),$$

which is $(1 - o(1)) \cdot n \log_d m_0$ since $m_0 = \omega(1)$. ■

4.2 Upper Bounds

In this subsection, we describe probabilistic merging networks whose size and depth match the lower bounds established in the previous subsection, up to lower order terms. The structure of the networks is similar to that of Batcher's odd-even merging networks.

We assume that the input consists of two lists $a_0, \ldots, a_{m_0-1}$ and $a_{m_0}, \ldots, a_{n-1}$ sorted in ascending order. The result is a single sorted list contained in $a_0, \ldots, a_{n-1}$. The odd-even merging network is defined by the following algorithm.

(1) Recursively merge the "even" lists $a_0, a_2, a_4, \ldots$ and $a_{m_0}, a_{m_0+2}, a_{m_0+4}, \ldots$, and the "odd" lists $a_1, a_3, a_5, \ldots$ and $a_{m_0+1}, a_{m_0+3}, a_{m_0+5}, \ldots$.

(2) Perform comparison-exchanges between a_{2i-1} and a_{2i}, for $1 \leq i < n/2$.

The recursion in Step (1) ends when the first sublist has length 1, and the second sublist has length m_1/m_0. (We will ignore integrality concerns for ease of presentation.) At this point, an insertion network of size m_1/m_0 and depth $\log(m_1/m_0) + 1$ can be used to merge the two sublists. In our probabilistic merging network, we essentially follow the same algorithm, except that we end the recursion at an earlier stage. For the correctness proof of our construction, we need the following lemma.

Lemma 4.3 *Let X be a subset of size m chosen at random from a set of n elements. Then for $0 \leq k < m$, the element of rank k in X has an overall rank of $kn/m \pm O(n\sqrt{(\log n)/m})$, with probability at least $1 - 1/poly(n)$.*

Proof: Similar to the proof of Lemma 3.6. ■

The following theorem shows that probabilistic merging networks with polynomially small failure probability require only about half as many comparators as worst-case networks.

Theorem 4 *For $m_0 = \log^{\omega(1)} n$, there exists an explicit construction of an (m_0, m_1)-merging network with success probability at least $1 - 1/poly(n)$ that has a size of $(\frac{1}{4} + o(1)) \cdot n \log m_0$ and a depth of $(1 + o(1)) \cdot \log(\frac{n}{\sqrt{m_0}})$.*

Proof: Given a random input permutation to an (m_0, m_1)-merging network, Lemma 4.3 implies that, with probability at least $1 - 1/\text{poly}(n)$, (1) any element a_k with $0 \leq k < m_0$ has a rank between $kn/m_0 - O(n\sqrt{(\log n)/m_0})$ and $kn/m_0 + O(n\sqrt{(\log n)/m_0})$, and (2) any element a_k with $m_0 \leq k < n$ has a rank between $(k - m_0)n/m_1 - O(n\sqrt{(\log n)/m_1})$ and $(k - m_0)n/m_1 + O(n\sqrt{(\log n)/m_1})$. We will refer to this interval of ranks as the *span* of a_k.

Now consider any input permutation with the property that the rank of every value a_k is within its span. (A random input permutation satisfies this property, with probability at least $1 - 1/\text{poly}(n)$.) Let $c = \log(\sqrt{m_0}) + \Theta(\log \log n)$, and consider a pair of "even" and "odd" sublists that have to be merged in the cth level of recursion of odd-even merge. At this point, the length of the sublists has been reduced to $m_0/2^c$ and $m_1/2^c$, respectively, and for any two elements a_i and a_j in the same sublist, we have $|i - j| \geq 2^c$.

It can now be checked that the spans of any two elements taken from the shorter sublist are disjoint. Also, no span of any element from the longer sublist can overlap with the span of more than one element from the shorter sublist. Finally, it can be shown that the span of an element from the shorter sublist can overlap with the span of at most m_1/m_0 elements from the longer sublist.

Thus, in order to merge the two sublists, it suffices to insert each element a_i of the shorter sublist into the sorted list of those elements in the longer sublist whose span overlaps with the span of a_i. This can be done with $m_0 + m_1$ size and $\log(m_1/m_0) + 1$ depth using worst-case insertion networks. After this step, the network proceeds in the same fashion as odd-even merge, incurring an additional c in depth and, thus, at most $c(m_0 + m_1)/2 = (1/4 + o(1)) \cdot n \log m_0$ in size. ∎

Note that as an immediate consequence of Theorem 4, we can obtain a simple sorting network based on merging that has only about half the size and depth of Batcher's original construction.

We point out that our probabilistic merging networks are actually proper "subnetworks" of the corresponding odd-even merging networks, and that they could also be derived by a careful analysis and "pruning" of these networks. For the case $m_0 = m_1$, it suffices to simply remove the first half of the network, except for the very first level. (A similar observation was very recently used by Rüb [24] to show that a variant of odd-even merge sort achieves optimal parallel speed-up on random inputs under high load.) However, for lists of different lengths, such an analysis seems more complicated than that in the above proof. Finally, another related result was given by Sedgewick [25], who analyzed the number of exchanges performed by odd-even merging on a random input. (In contrast, we are interested in the "predictability" of the outcome of a comparison-exchange, rather than in what this outcome is.)

4.3 Small-Depth Probabilistic Sorting Networks

In this subsection, we describe two probabilistic sorting networks based on ideas related to higher-order merging. The first network has an extremely simple structure, and achieves a depth of $O(\log^{1+\varepsilon} n)$, for any $\varepsilon > 0$. The second network is only slightly more complicated, and has a depth of $O(\log n \log \log n)$. Both of the networks have a success probability at least $1 - 1/\text{poly}(n)$.

Although the depth of our networks does not match that of the asymptotically optimal networks of Ajtai, Komlós, and Szemerédi [2] and Leighton and Plaxton [16], we believe that our networks are of interest due to their simplicity. In this context, we point out that all (worst-case and probabilistic) sorting networks of depth $o(\log^2 n)$ previously known [2, 16, 22] have a significantly more complicated structure and/or proof of correctness.

We describe our networks as sorting algorithms operating on an array of n elements, to be sorted into ascending order. The algorithms make use of an operation called (m, k)-*sort-and-shuffle*, which operates on an array of m elements by first sorting the elements in each block of size k, and then shuffling the m/k sorted lists together. (That is, the element in position i of block j is moved to position $i \cdot m/k + j$ of the array, for $0 \leq i < k$ and $0 \leq j < m/k$.) The next lemma can be proved in a fachion similar to that of Lemma 3.6.

Lemma 4.4 *After an (m, k)-sort-and-shuffle has been applied to a random input permutation, every element is within $O(m\sqrt{T/k})$ of its final destination with* *probability at least* $1-e^{-\Theta(T)}$, *where* $T \geq c\log m$ *for some sufficiently large constant* c.

The first algorithm can be seen as a straightforward higher-order merge, and operates as follows.

(1) Perform an (n, n^{δ})-sort-and-shuffle, for some constant $\delta > 0$. It follows from Lemma 4.4 that every element is within $n^{1-\delta/3}$ of its destination, with probability at least $1-\exp(-\Theta(n^{\delta/3}))$.

(2) Perform a "random" permutation in each block of size $n^{1-\delta/3}$. Then sort each block.

(3) Perform two steps of merging between adjacent blocks to complete the sort.

We can implement this algorithm as a probabilistic sorting network as follows. We use recursion to sort the blocks in Steps (1) and (2); when the size of the blocks becomes $2^{\sqrt{\log n}}$, we apply bitonic sort [7]. (Here, we terminate the recursion early to ensure a satisfactory success probability of the whole network.) The "random" permutation before Step (2) is needed to prevent dependencies in the recursive sorting in Step (2), which would complicate the analysis. Of course, a sorting network cannot perform a random permutation in the strict sense. However, the fact that the algorithm works correctly if we apply a random permutation before Step (2) implies the existence of a fixed permutation such that the sorting network works correctly on a random input permutation. Thus, the above description is non-constructive.

The depth of the network is given by the relations $D(n) \leq D(n^{\delta}) + D(n^{1-\delta/3}) + O(\log n)$ and $D(2^{\sqrt{\log n}}) = O(\log n)$. By the substitution method, we can verify that $D(n) = O((\log n)^{1+\log_{1/\delta} 3})$. Note that we can obtain depth $O(\log^{1+\varepsilon} n)$ for any $\varepsilon > 0$, by choosing δ sufficiently small.

Our second network has depth $O(\log n \log\log n)$, and can be described by the following algorithm:

(1) Perform an $(n, \log^3 n)$-sort-and-shuffle. It follows from Lemma 4.4 that every element is within $\frac{n}{\log n}$ of its destination with probability at least $1 - 1/\text{poly}(n)$.

(2) Partition the array into blocks of size $\frac{2n}{\log n}$, and apply a "random" permutation in each block. Then perform a $(\frac{2n}{\log n}, \log^3 n)$-sort-and-shuffle in each block. Repeat this procedure, using a second partition of the array into blocks of size $\frac{2n}{\log n}$ in which the boundaries of the blocks are shifted by $\frac{n}{\log n}$ positions. Afterwards, it can be shown by Lemma 4.4 that every element is within $\frac{n}{\log^2 n}$ of its destination, with probability at least $1 - 1/\text{poly}(n)$.

(3) Continue in this fashion, at each step reducing the block size by a factor of $\log n$, and using two sets of overlapping blocks.

The sorting of blocks of size $\log^3 n$ in the sort-and-shuffle operations can be done using bitonic sort. Thus, each of the $O(\log n/\log\log n)$ iterations has depth $O((\log\log n)^2)$, and the complete network has depth $O(\log n \log\log n)$. The analysis of the network is simple and is omitted in this paper.

5 Concluding Remarks

In this paper, we have proved tight bounds (up to constant factors or lower order terms) on the size of probabilistic networks for selection and merging, and described two simple small-depth probabilistic sorting networks.

An obvious open question is to determine the leading constants in the optimal size and depth bounds for selection and sorting networks. This seems to be a very difficult problem, as these constants are unknown even for the (more extensively studied) case of worst-case networks. The readers are referred to [1, 11, 16] for the best currently known bounds for worst-case and probabilistic sorting networks, and to [5, 9, 26] for the best currently known bounds for worst-case selection networks (we are not aware of any depth upper bound for worst-case $(n, n/2)$-selection networks that is better than the bound for worst-case sorting networks).

Another open question is whether the results of this paper can be extended to the class of *randomized* comparator networks introduced in [16]. (These networks output the correct result with high probability on any input permutation, and make use of randomized switching elements that exchange two input values with probability 1/2.) Clearly, any upper bound for randomized comparator networks implies an identical (but not necessarily constructive) upper bound for probabilistic networks, and hence any lower bound for probabilistic networks also extends to randomized networks.

In the case of merging, it is easy to see that the nearly tight worst-case lower bound in [18] also extends to randomized networks. For sorting, the best randomized upper and lower bounds known are given by the results in [16] and [11], respectively. Finally, for selection, the results of this paper imply a size lower bound of $\Omega(n \log \log n)$, while the best upper bound known is $O(n \log n)$.

Finally, it remains an important open problem to find a truly simple (worst-case or probabilistic) sorting network of depth $O(\log n)$.

Acknowledgement

We are grateful to Greg Plaxton for many stimulating discussions. We thank Endre Szemerédi for pointing out an error in an early proof of Lemma 3.7, and for providing a fix to that proof by using a result from [4]. We also thank the anonymous referees for valuable comments.

References

[1] M. Ajtai, J. Komlós, M. S. Paterson, and E. Szemerédi. k-sorting. In preparation.

[2] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the 15th Annual ACM Symposium on the Theory of Computing*, pages 1–9, May 1983.

[3] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1–19, 1983.

[4] M. Ajtai, J. Komlós, and E. Szemerédi. Halvers and expanders. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 686–689, October 1992.

[5] V. E. Alekseyev. Sorting algorithms with minimum memory. *Kibernetika*, 5:99–103, 1969.

[6] N. Alon and J. H. Spencer. *The Probabilistic Method.* Wiley-Interscience, New York, NY, 1991.

[7] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 32, pages 307–314, 1968.

[8] T. Ikeda. A probabilistic selection network with butterfly networks. In *ISAAC '93, Lecture Notes in Computer Science*, volume 762, pages 267–276, 1993.

[9] M. Jimbo and A. Maruoka. Selection networks with $8\,n \log n$ size and $O(\log n)$ depth. In *ISAAC '92, Lecture Notes in Computer Science*, volume 650, pages 165–174, 1992.

[10] N. Kahale. Eigenvalues and expansion of regular graphs. Technical Report 93-70R, DIMACS, Rutgers University, 1993.

[11] N. Kahale, T. Leighton, Y. Ma, C. G. Plaxton, T. Suel, and E. Szemerédi. Lower bounds for sorting networks. In *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing*, May 1995. To appear.

[12] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley, Reading, MA, 1973.

[13] E. A. Lamagna. The complexity of monotone networks for certain bilinear forms, routing problems, sorting, and merging. *IEEE Trans. Comput.*, 28:773–782, 1979.

[14] T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes.* Morgan-Kaufmann, San Mateo, CA, 1992.

[15] T. Leighton, Y. Ma, and C. G. Plaxton. Highly fault-tolerant sorting circuits. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 458–469, October 1991.

[16] T. Leighton and C. G. Plaxton. A (fairly) simple circuit that (usually) sorts. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 458–469, October 1990.

[17] Y. Ma. *Fault-Tolerant Sorting Networks.* PhD thesis, MIT, 1994.

[18] P. B. Miltersen, M. S. Paterson, and J. Tarui. The asymptotic complexity of merging networks. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 236–246, October 1992.

[19] N. Pippenger. Communication networks. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 805–833. North-Holland, Amsterdam, 1990.

[20] N. Pippenger. Selection networks. *SIAM J. Comput.*, 20:878–887, 1991.

[21] N. Pippenger and L. G. Valiant. Shifting graphs and their applications. *J. Assoc. Comput. Mach.*, 23:423–432, 1976.

[22] C. G. Plaxton. A hypercubic sorting network with nearly logarithmic depth. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 405–416, May 1992.

[23] C. Rüb. Lower bounds for merging on the hypercube. In *Proceedings of the 2nd Italian Conference on Algorithms and Complexity, Spinger LNCS 778*, pages 213–222, 1994.

[24] C. Rüb. On the average running time of odd-even merge sort. In *Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science, Spinger LNCS 900*, pages 491–502, February 1995.

[25] R. Sedgewick. Data movement in odd-even merging. *SIAM J. Comput.*, 7:239–272, 1978.

[26] A. C. Yao. Bounds on selection networks. *SIAM J. Comput.*, 9:566–582, 1980.

[27] A. C. Yao and F. F. Yao. Lower bounds on merging networks. *J. Assoc. Comput. Mach.*, 23:423–432, 1976.