A (fairly) Simple Circuit that (usually) Sorts

Tom Leighton[1,2] *C. Greg Plaxton*[1]

[1]Laboratory for Computer Science and
[2]Mathematics Department
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

This paper provides an analysis of a natural k-round tournament over $n = 2^k$ players, and demonstrates that the tournament possesses a surprisingly strong ranking property. The ranking property of this tournament is exploited by using it as a building block for efficient parallel sorting algorithms under a variety of different models of computation. Three important applications are provided. First, a sorting circuit of depth $7.44 \log n$ is defined that sorts all but a superpolynomially small fraction of the $n!$ possible input permutations. Second, a randomized sorting algorithm is given for the hypercube and related parallel computers (the butterfly, cube-connected cycles and shuffle-exchange) that runs in $O(\log n)$ word steps with very high probability. Third, a randomized algorithm is given for sorting n $O(m)$-bit records on an $n \log n$ node butterfly that runs in $O(m + \log n)$ bit steps with very high probability.

1 Introduction

Consider the following k-round tournament defined over $n = 2^k$ players. In the first round, $n/2$ matches are played according to a *random* pairing of the n players. The next $k-1$ rounds are defined by recursively running a tournament amongst the $n/2$ winners, and (in parallel) a separate tournament amongst the $n/2$ losers. Note that the depth k comparator circuit corresponding to this tournament is an n-input butterfly network in which the input is a random permutation and the two outputs of each comparator gate are oriented in the same direction. Hence, this tournament will be referred to as the *butterfly tournament* of order k.

After the tournament has been completed, each player has achieved a unique sequence of match outcomes (wins and losses, 1's and 0's) of length k. Let player i be the player that achieves a W-L sequence corresponding to the k-bit number i, that is, the player "routed" to the ith output of the n-input butterfly comparator circuit, $0 \leq i < n$.[1] Assume that the outcomes of all matches are determined by an underlying total order. Further assume that the tournament has available n distinct amounts of prize money to be assigned to the n possible outcome sequences. How should these amounts be assigned? Clearly the largest amount of money should be assigned to player $n-1 = W^k$, who is guaranteed to be the best player. Similarly, the smallest prize should be awarded to player $0 = L^k$. On the other hand, it is not clear how to rank all of the remaining $n-2$ W-L sequences. For instance, in the case $n = 2^8$, should the sequence WLWLLWLL be rated above or below the sequence LLLWWWWW? Intuition and standard practice say that the player with the 5–3 record should be ranked above the player with the 3–5 record. As we will show in Section 3, however, this is not true in this example. In fact, we will see that the standard practice of matching and ranking players based on numbers of wins and losses is not very good. Rather, we will see that it is better to match and rank players based on their precise sequences of previous wins and losses.

This research was supported by an NSERC postdoctoral fellowship, the Defense Advanced Research Projects Agency under Contracts N00014–87–K–825 and N00014–89–J–1988, the Air Force under Contract AFOSR–89–0271, and the Army under Contract DAAL–03–86–K–0171.

[1]The W-L sequences should be read from left to right, that is, the butterfly is oriented in such a way that the most significant bit of the output position is determined by the first comparison.

The analysis of Section 3 not only shows that WLWLLWLL is a better record than LLLWWWWW, but also provides an efficient algorithm for computing a fixed permutation π of the set $\{0, \ldots, n-1\}$ such that with extremely high probability, the actual rank of all but a small, fixed subset of the players is well-approximated by $\pi(i)$, $0 \le i < n$. See Theorem 1 for a precise formulation of this result. Furthermore, by modifying the basic algorithm it is possible to construct a k-round tournament that well-approximates everyone.[2]

Why might one suspect that the butterfly tournament would admit such a strong ranking property? Intuitively, a comparison will yield the most information if it is made between players expected to be of approximately equal strength; the outcome of a match between a player whose previous record is very good and one whose previous record is very bad is essentially known in advance and hence will normally provide very little information. The butterfly tournament has the property that when two players meet in the ith round, they have achieved the same sequence of outcomes in two independent butterfly tournaments T_0 and T_1 of order $i-1$. By symmetry, exactly half of the $n!$ possible input permutations will lead to a win by the player representing T_0, and half will lead to a win by the player representing T_1.

In Sections 4 and 5, the strong ranking property of the butterfly tournament is used to build efficient parallel sorting algorithms under a variety of different computational models. Some of our results are probabilistic in nature, and the following convention will be adopted in order to distinguish between the three levels of "high probability" that arise. The phrases *with high probability*, *with very high probability*, and *with extremely high probability* will be applied to events that fail to occur with probability $O(n^{-c})$, $O(2^{-2^{c\sqrt{\log n}}})$, and $O(2^{-n^c})$, respectively, where c is some positive constant and n is the input size.

Three significant applications of the butterfly tournament are presented. In Section 4, a comparator circuit of depth $7.44 \log n$ is defined that sorts a randomly chosen input permutation with very high probability. At the expense of allowing the circuit to fail on a very small fraction of the $n!$ possible input permutations, this construction improves upon the asymptotic depth of the best previously known sorting circuits by several orders of magnitude [2][7]. Furthermore, the topology of our circuit is quite simple; it is closely related to that of a butterfly and does not rely on expanders.

In Section 5.3, a randomized sorting algorithm is given for the hypercube and related parallel computers (the butterfly, cube-connected cycles and shuffle-exchange) that runs in $O(\log n)$ word steps with very high probability. A number of previous randomized sorting algorithms exist for these networks. The Flashsort algorithm of Reif and Valiant [9], defined for the cube-connected cycles, also achieves optimal $O(\log n)$ time, although the algorithm makes use of an $O(\log n)$-sized priority queue at each processor. A similar result with constant size queues is described by Leighton, Maggs, Ranade and Rao [6]. Like Batcher's $O(\log^2 n)$ bitonic sorting algorithm, our sorting algorithm is *non-adaptive* in the sense that it can be described solely in terms of oblivious routing and compare-interchange operations; there is no queueing. Also, the probability of success of our algorithm is *very high*, which represents an improvement over the high probability level achieved in [6] and [9].

Our third and final application is described in Section 5.4, where we give a randomized algorithm for sorting n $O(m)$-bit records on an $n \log n$ node butterfly that runs in $O(m + \log n)$ bit steps with very high probability. This is a remarkable result in the sense that the time required for sorting is shown to be no more than a constant factor larger than the time required to examine a record. The only previous result of this kind that does not rely on the AKS sorting circuit is the recent work of Aiello, Leighton, Maggs and Newman, which provides a randomized bit-serial *routing* algorithm that runs in optimal time with high probability on the hypercube [1]. That paper does not address either the combining or sorting problems, however, and does not apply to any of the bounded-degree variants of the hypercube. All previously known algorithms for routing and sorting on bounded degree variants of the hypercube, and for sorting on the hypercube, require $\Omega(\log^2 n)$ bit steps.

2 Preliminaries

Let $B(n,\ p,\ k) = \binom{n}{k} p^k (1-p)^k$ denote the probability of obtaining exactly k heads on n independent coin tosses where each coin toss yields a head with probability p, $0 \le p \le 1$. We will make use of the following fact:

$$B(n, k/n, k) = \Omega(1/\sqrt{n}). \tag{1}$$

Throughout this paper, the "log" function refers to the base 2 logarithm.

[2] This result is not difficult to work out given the material in Section 3, but we have deferred the details to the final version of the paper.

Let $\text{bin}(i,k)$ denote the k-bit binary string corresponding to the integer i, $0 \le i < 2^k$.

3 Tournament Analysis

In this section it will be proven that the butterfly tournament defined in Section 1 has a strong ranking property. The proof relies on the construction of a fixed permutation π such that the actual rank of player i is well-approximated by $\pi(i)$ for all but a small number of values of i, $0 \le i < n$. Recall that player i is the unique player whose W-L sequence corresponds to the $\log n$-bit binary representation of the integer i. Formally, the following result will be established, with $\gamma \approx 0.822$.

Theorem 1 Let $n = 2^k$ where k is some nonnegative integer, and let $X = \{0, \ldots, n-1\}$. Then there exists a fixed permutation π of X, a positive constant γ strictly less than unity, and a fixed subset Y of X such that $|Y| = O(n^\gamma)$ and the following statement holds true with extremely high probability: If n players participate in a butterfly tournament, then the actual rank of player i lies in the range $[\pi(i) - O(n^\gamma), \pi(i) + O(n^\gamma)]$ for all i in $X \setminus Y$.

Furthermore, an efficient algorithm will be given for computing the subset Y and permutation π mentioned in the theorem.

The zero-one principle for sorting circuits states that an n-input (and hence, n-output) comparator circuit is a sorting circuit if and only if it correctly sorts all 2^n 0-1 inputs [5]. Our analysis of the butterfly tournament makes use of a simple probabilistic generalization of the zero-one principle.

Given an n-input comparator circuit, let $f_i(k)$ denote the probability that the ith output is a 0 when the input is a randomly chosen permutation of k 0's and $n-k$ 1's, $0 \le i < n$, $0 \le k \le n$. It is straightforward to prove that $f_i(k)$ is a monotonically nondecreasing function of k.

By the aforementioned zero-one principle, a comparator circuit is a sorting circuit if and only if

$$f_i(k) = \begin{cases} 1 & \text{if } k > i \\ 0 & \text{otherwise} \end{cases}$$

for $0 \le i < n$, $0 \le k \le n$.

Lemma 3.1 Suppose that the ith output of a comparator circuit C satisfies $f_i(u) \le \epsilon$ and $f_i(v) \ge 1 - \epsilon'$. Then on a random input permutation of $\{0, \ldots, n-1\}$ the ith output of C will have rank k in the range $u \le k < v$ with probability at least $1 - \epsilon - \epsilon'$.

Proof: The ith output has rank strictly less than u with probability ϵ, and has rank strictly less than v with probability ϵ'. The claim follows. □

Thus, a sharp threshold result for the f_i's corresponding to a particular circuit C will establish a strong average case sorting property for C. For technical reasons, it will be convenient for us to consider a slightly different set of output probability functions. Given an n-input comparator circuit, let $g_i(p)$ denote the probability that the ith output is a 0 when each input is independently set to 0 with probability p, and to 1 with probability $1-p$. Here p is a real value in $[0,1]$. It is easy to verify that the g_i's must satisfy the following properties: $g_i(0) = 0$, $g_i(1) = 1$, and $g_i'(p) > 0$, $0 \le p < 1$. Furthermore, g_i can be written in terms of f_i as follows:

$$g_i(p) = \sum_{0 \le k \le n} B(n,\ p,\ k) f_i(k). \tag{2}$$

The following lemma proves a threshold result for the g_i's that is analogous to Lemma 3.1.

Lemma 3.2 Suppose that the ith output of an n-input comparator circuit C satisfies $g_i(u) \le 2^{-n^\epsilon}$ and $g_i(v) \ge 1 - 2^{-n^{\epsilon'}}$. Then on a random input permutation of $\{0, \ldots, n-1\}$ the ith output of C will have rank k in the range $\lfloor un \rfloor \le k < \lceil vn \rceil$ with extremely high probability.

Proof: By Equation 2, $g_i(k/n) \ge B(n, k/n, k) f_i(k)$. Thus, Equation 1 implies that $f_i(k) = O(\sqrt{n} g_i(k/n))$, and hence that $f_i(\lfloor un \rfloor) = O(\sqrt{n} g_i(\lfloor un \rfloor / n)) = O(\sqrt{n} g_i(u))$. A symmetric argument can be used to show that $f_i(\lceil vn \rceil)$ is exponentially close to 1. The claim follows by Lemma 3.1. □

We now turn to the analysis of the butterfly tournament. For convenience, we adopt a slightly different notation for the g_i's. In particular, the function $g_i(p)$ corresponding to the ith output of an $n = 2^k$-input butterfly tournament will be denoted $a_\alpha(p)$ where $\alpha = \text{bin}(i,k)$. It is straightforward to prove that the a_α's are polynomials of degree $2^{|\alpha|}$ that can be constructed inductively as follows:

$$\begin{aligned} a_\phi(p) &= p, \\ a_{\alpha 0}(p) &= 2a_\alpha(p) - a_\alpha(p)^2, \\ a_{\alpha 1}(p) &= a_\alpha(p)^2. \end{aligned}$$

Our goal is to prove a sharp threshold result for the polynomials $a_\alpha(p)$ corresponding to all but $O(n^\gamma)$ of the n distinct strings α of length $\log n$, for some positive constant γ less than 1.

In order to prove a sharp threshold result for some polynomial $a_\alpha(p)$, we will need to show that for some p, $a_\alpha(p-n^{-\epsilon}) < 2^{-n^\delta}$ and that $a_\alpha(p+n^{-\epsilon}) > 1-2^{-n^\delta}$ for some constants $\delta, \epsilon > 0$. To accomplish this task, it will be useful to calculate an inverse function of a_α. Namely, we define $b_\alpha(z)$ to be the value of p for which $a_\alpha(p) = z$. In other words, $a_\alpha(b_\alpha(z)) = z$ for all z, $0 \le z \le 1$.

Of particular interest are the values

$$\begin{aligned} u_\alpha &= b_\alpha(2^{-n^\delta}), \\ p_\alpha &= b_\alpha(1/2), \text{ and} \\ v_\alpha &= b_\alpha(1-2^{-n^\delta}), \end{aligned}$$

where $n = 2^{|\alpha|}$ and δ is some small positive constant to be specified later. The value of p_α is interesting because we will expect the rank of player i to be close to $p_{\mathrm{bin}(i,k)}n$ where $k = \log n$. More precisely, we know by Lemma 3.2 that the rank of the player with record α will be between $\lfloor u_\alpha n \rfloor$ and $\lceil v_\alpha n \rceil$ with probability at least $1-2^{-n^\delta+1}$. Since $u_\alpha < p_\alpha < v_\alpha$ for all α, this means that the rank of the player with record α will be $p_\alpha n$ to within a $\pm$ error of $(v_\alpha - u_\alpha)n$ positions with extremely high probability. To prove Theorem 1, it will thus suffice to show that $v_\alpha - u_\alpha = O(n^{\gamma-1})$ for all but $O(n^\gamma)$ strings α. This is because $v_\alpha - u_\alpha = O(n^{\gamma-1})$ implies that the rank of player α is $\lfloor p_\alpha n \rfloor$ up to a $\pm$ error of $O(n^\gamma)$ with extremely high probability.

To be completely precise, we should point out that the values of $\lfloor p_\alpha n \rfloor$ are not all distinct. Hence, it is not entirely legitimate to define $\pi(i) = \lfloor p_{\mathrm{bin}(i,k)} n \rfloor$. However, this technicality can be easily dealt with by sorting the p_α's and setting $\pi(i)$ to the rank achieved by $\lfloor p_{\mathrm{bin}(i,k)} n \rfloor$. A simple argument reveals that the resulting total order correctly estimates the rank of all but $O(n^\gamma)$ players to within $O(n^\gamma)$ positions with extremely high probability.

The hard part, of course, is to prove that $v_\alpha - u_\alpha = O(n^{\gamma-1})$ for all but $O(n^\gamma)$ strings α. This task will be greatly simplified by the fact that the inverse polynomials $b_\alpha(z)$ can be constructed in an analogous (but reverse) manner from the $a_\alpha(p)$'s. In particular,

$$\begin{aligned} b_\phi(z) &= z, \\ b_{0\alpha}(z) &= 1-\sqrt{1-b_\alpha(z)}, \\ b_{1\alpha}(z) &= \sqrt{b_\alpha(z)}. \end{aligned} \tag{3}$$

In other words, the polynomial $b_\alpha(z)$ is constructed by reversing and inverting the operations performed to construct $a_\alpha(p)$, so that if we apply a_α to $b_\alpha(z)$, we are left with z.

Although the $b_\alpha(z)$ are not polynomials, they are still fairly easy to work with. For example, $b_\alpha(z)$ is strictly increasing for all α, and

$$b_{\beta\alpha}(z) = b_\beta(b_\alpha(z)) \tag{4}$$

for all α and β. We can also easily compute the values of u_α, p_α and v_α from the recurrences in Equation 3. For example, given the strings

$$\alpha = \text{WLWLLWLL and } \beta = \text{LLLWWWWW}$$

mentioned in the introduction, we can apply the recurrences in Equation 3 to determine that

$$p_\alpha = 0.563 \text{ and } p_\beta = 0.619.$$

Hence, player α should be ranked higher than player β even though player β has a better record (5–3 vs. 3–5)! This example illustrates the fact that early wins are much more important than later wins in computing ranks, a fact often overlooked when designing tournaments.

As the number of players n grows large, it is possible to find even more striking examples of this phenomenon. For example, the player who wins his first $(\log n)/3$ matches and then loses the rest will be among the best $n^{1-\epsilon}$ players with extremely high probability, while the player who loses his first $(\log n)/3$ matches and then wins the rest will be among the worst $n^{1-\epsilon}$ players with extremely high probability (for some $\epsilon > 0$). This is notwithstanding the fact that the "lesser" player won twice as many matches as the "better" player. (These facts are not too difficult to prove given the techniques in this paper, but we will not go through the analysis here.) Such examples also illustrate the fact that tournaments that match and rank players by the number of wins and losses (as is common) are poorly designed. As we show in this paper, it is much better to arrange matches based on the exact sequence of previous wins and losses.

In order to show that $u_\alpha = b_\alpha(2^{-n^\delta})$ and $v_\alpha = b_\alpha(1-2^{-n^\delta})$ are very close for all but a few α, it is useful to analyze how the "distance" between $p = 2^{-n^\delta}$ and $q = 1-2^{-n^\delta}$ decreases as the recurrences in Equation 3 are applied to p and q to form $u_\alpha = b_\alpha(p)$ and $v_\alpha = b_\alpha(q)$. To measure the distance between two values $p < q$, we will use the function

$$\Delta(p,q) = \log \frac{q(1-p)}{(1-q)p}.$$

Since $q > p$ and $x/(1-x)$ is an increasing function, $\Delta(p,q)$ is always positive.

At the start, we have $\Delta(2^{-n^\delta}, 1-2^{-n^\delta}) \sim 2n^\delta$, which reflects the fact that 2^{-n^δ} and $1-2^{-n^\delta}$ are very far apart. At the end, we want $b_\alpha(p)$ and $b_\alpha(q)$ to be very close, which will be enforced if $\Delta(b_\alpha(p), b_\alpha(q)) \sim n^{\gamma-1}$. More precisely, simple calculus shows that for any $y > x$,

$$y - x \leq \Delta(x, y).$$

Hence, we will want to prove that $h_\alpha(2^{-n^\delta}, 1 - 2^{-n^\delta}) \leq n^{\gamma-1-\delta}$ for all but $O(n^\gamma)$ strings α, where

$$h_\alpha(p, q) = \frac{\Delta(b_\alpha(p), b_\alpha(q))}{\Delta(p, q)}.$$

Once this is done, we will have proved Theorem 1 since $h_\alpha(2^{-n^\delta}, 1 - 2^{-n^\delta}) \leq n^{\gamma-1-\delta}$ implies

$$\begin{aligned} v_\alpha - u_\alpha &\leq \Delta(u_\alpha, v_\alpha) \\ &= \Delta(b_\alpha(2^{-n^\delta}), b_\alpha(1 - 2^{-n^\delta})) \\ &= h_\alpha(2^{-n^\delta}, 1 - 2^{-n^\delta})\Delta(2^{-n^\delta}, 1 - 2^{-n^\delta}) \\ &\leq 2n^{\gamma-1-\delta} n^\delta \\ &= 2n^{\gamma-1}. \end{aligned}$$

The remainder of the proof focusses on showing that for any $p < q$, $h_\alpha(p, q)$ is small for all but a few strings α. The first step in this process is to observe that

$$\begin{aligned} h_\phi(p, q) &= 1, \\ h_{0\alpha}(p, q) &= h_0(b_\alpha(p), b_\alpha(q))h_\alpha(p, q), \text{ and} \\ h_{1\alpha}(p, q) &= h_1(b_\alpha(p), b_\alpha(q))h_\alpha(p, q). \end{aligned} \tag{5}$$

These identities follow directly from the definition of $h_\alpha(p, q)$ and Equation 4 (with $\beta = 0$ and $\beta = 1$).

If it were true that there was a constant $\rho < 1$ such that $h_0(x, y) < \rho$ and $h_1(x, y) < \rho$ for all x, y, we would now be done, since we could repeatedly apply the recurrences of Equation 5 to show that $h_\alpha(p, q) \leq \rho^{\log n} = n^{-\log(1/\rho)}$ for all p, q and α. Unfortunately, this is not the case. In fact, it is not even true that $h_\alpha(2^{-n^\delta}, 1 - 2^{-n^\delta})$ is small for all α. However, it is true that $h_0(x, y)$ and $h_1(x, y)$ are very often small, and we can achieve nearly the same effect by using a potential function argument. In particular, we will use the potential function

$$H_\lambda(k, p, q) = \sum_{0 \leq i < 2^k} \left[h_{\mathrm{bin}(i,k)}(p, q)\right]^\lambda.$$

In what follows we show how to upper bound $H_\lambda(k, p, q)$ in terms of a constant

$$r^*_\lambda = \limsup_{0<x<y<1} \left\{h_0(x, y)^\lambda + h_1(x, y)^\lambda\right\}$$

that will play a role similar to the role played by ρ in the preceding paragraph.

Lemma 3.3 For all nonnegative integers k and real values p, q and λ such that $0 < p < q < 1$ and $\lambda > 1$,

$$H_\lambda(k, p, q) \leq (r^*_\lambda)^k.$$

Proof: The proof is by induction on k. The base case, $k = 0$, is trivial since $h_\phi(p, q) = 1$. For $k > 0$ note that for any binary string α of length $k - 1$,

$$\begin{aligned} h_{0\alpha}(p, q)^\lambda + h_{1\alpha}(p, q)^\lambda &\leq r^*_\lambda h_\alpha(p, q)^\lambda \\ &\leq (r^*_\lambda)^k, \end{aligned}$$

by the definition of r^*_λ, the recurrences in Equation 5, and the inductive hypothesis. □

The following lemma shows how the upper bound on the potential function can be used to upper bound the number of strings α for which $h_\alpha(p, q)$ is too large.

Lemma 3.4 For any fixed choice of real values p, q and λ such that $0 < p < q < 1$ and $\lambda > 1$, the inequality

$$h_\alpha(p, q) > n^{\beta-1}$$

is satisfied by at most n^β of the n binary strings α of length $k = \log n$, where

$$\beta = \frac{\log r^*_\lambda + \lambda}{1 + \lambda}.$$

Proof: Let λ be any fixed real value. If there exist n^β binary strings of length k such that $h_\alpha(p, q) > n^{\beta-1}$ then

$$H_\lambda(k, p, q) > n^\beta n^{\lambda(\beta-1)}.$$

The inequality of Lemma 3.3 implies that this is not possible if $\beta > (\log r^*_\lambda + \lambda)/(1 + \lambda)$. □

At this point, it remains only to find a value of $\lambda > 1$ for which $\beta = (\log r^*_\lambda + \lambda)/(1 + \lambda)$ is small. Unfortunately, this is a fairly messy task. As it turns out, if $\lambda = 3.609$, then $r^*_\lambda < 1.133$ and $\beta < 0.822$. Given these values, we can prove Theorem 1 with $\gamma = 0.822$. Recall that $X = \{0, \ldots, n-1\}$ where $n = 2^k$. Let Y denote that subset of X containing all k-bit binary strings α such that

$$h_\alpha(2^{-n^\delta}, 1 - 2^{-n^\delta}) > n^{-0.178},$$

where δ is a sufficiently small positive constant. Lemma 3.4 implies that $|Y| = O(n^{0.822})$. By the preceding analysis, we know that the rank of every $i \in X \setminus Y$ is within $O(n^{0.822})$ of $\pi(i)$ with extremely high probability.

Except for the matter of showing $r^*_\lambda < 1.133$ for $\lambda = 3.609$, we have now completed the proof of Theorem 1. In what follows, we describe methods for upper bounding r^*_λ. We start with a general purpose lemma.

Lemma 3.5 Let I denote an arbitrary real interval and let f_0, f_1 and f_2 each denote a strictly increasing continuous and differentiable function over I. Let

$$f_3(x,y,\lambda) = \left(\frac{f_0(y)-f_0(x)}{f_2(y)-f_2(x)}\right)^\lambda + \left(\frac{f_1(y)-f_1(x)}{f_2(y)-f_2(x)}\right)^\lambda$$

where $x, y \in I$ and λ is a real value strictly greater than unity. Then for all x, y in I,

$$f_3(x,y,\lambda) \le \max_{x \in I} f_3(x,x,\lambda).$$

Proof: Note that because f_2 is strictly increasing and differentiable, l'Hopital's rule implies that $f_3(x,y,\lambda)$ is well-defined even if $x = y$.

It is sufficient to prove that given any pair of real values x and y such that $x < y$, then there exists a value w in (x,y) such that either $f_3(x,w,\lambda) > f_3(x,y,\lambda)$ or $f_3(w,y,\lambda) > f_3(x,y,\lambda)$. To prove this, choose w so that $f_2(w) - f_2(x) = f_2(y) - f_2(w)$, and let $s_0 = f_0(w) - f_0(x)$, $s_1 = f_0(y) - f_0(w)$, $t_0 = f_1(w) - f_1(x)$, $t_1 = f_1(y) - f_1(w)$, and $u = f_2(w) - f_2(x) = f_2(y) - f_2(w)$. Note that s_0, s_1, t_0, t_1, and u are all strictly positive. Then

$$\begin{aligned} f_3(x,w,\lambda) &= \left(\frac{s_0}{u}\right)^\lambda + \left(\frac{t_0}{u}\right)^\lambda, \\ f_3(w,y,\lambda) &= \left(\frac{s_1}{u}\right)^\lambda + \left(\frac{t_1}{u}\right)^\lambda, \text{ and} \\ f_3(x,y,\lambda) &= \left(\frac{s_0+s_1}{2u}\right)^\lambda + \left(\frac{t_0+t_1}{2u}\right)^\lambda. \end{aligned}$$

For $\lambda > 1$, the function z^λ is strictly convex, so

$$\begin{aligned} \left(\frac{s_0}{u}\right)^\lambda + \left(\frac{s_1}{u}\right)^\lambda &> 2\left(\frac{s_0+s_1}{2u}\right)^\lambda, \text{ and} \\ \left(\frac{t_0}{u}\right)^\lambda + \left(\frac{t_1}{u}\right)^\lambda &> 2\left(\frac{t_0+t_1}{2u}\right)^\lambda. \end{aligned}$$

Summing these inequalities, we find that $f_3(x,w,\lambda) + f_3(w,y,\lambda) > 2f_3(x,y,\lambda)$, which implies the desired result. □

Lemma 3.6 For all $\lambda > 1$,

$$r_\lambda^* = \max_{0 \le z \le 1} \left[\left(\frac{1+\sqrt{z}}{2}\right)^\lambda + \left(\frac{1+\sqrt{1-z}}{2}\right)^\lambda \right].$$

Proof: The first step is to apply Lemma 3.5 with $I = (0,1)$, $f_0(z) = \log[b_0(z)/(1-b_0(z))]$, $f_1(z) = \log[b_1(z)/(1-b_1(z))]$, and $f_2(z) = \log[z/(1-z)]$. Then $\Delta(b_0(x), b_0(y)) = f_0(x) - f_0(y)$, $\Delta(b_1(x), b_1(y)) = f_1(x) - f_1(y)$, and $\Delta(x,y) = f_2(x) - f_2(y)$. Hence $f_3(x,y,\lambda) = h_0(x,y)^\lambda + h_1(x,y)^\lambda$, and we know from Lemma 3.5 that the limiting value of $h_0(x,y)^\lambda + h_1(x,y)^\lambda$ is obtained for $x \sim y$. Using l'Hopital's rule and elementary calculus, it can be shown that

$$\lim_{\epsilon \to 0} h_0((1-\epsilon)y, y) = \frac{1+\sqrt{1-y}}{2},$$

and that

$$\lim_{\epsilon \to 0} h_1((1-\epsilon)y, y) = \frac{1+\sqrt{y}}{2}.$$

This completes the proof. □

For $\lambda = 3$, we can use elementary calculus to show that $r_\lambda^* = (10 + 7\sqrt{2})/16$ (which is attainable for $z = 1/2$). This results in a value of $\beta < 0.829$. Using numerical calculations, we have determined that for $\lambda = 3.609$, $r_\lambda^* < 1.133$ and that $\beta < 0.822$. We suspect that this is essentially the best constant obtainable by this method.

4 A Sorting Circuit

Given Theorem 1, it is now a relatively simple task to design an $O(\log n)$ depth circuit that sorts a random input with very high probability. The transformation consists of two basic components, outlined below:

1. A procedure for converting the network of Theorem 1 that approximately computes the rank of i for i in $X \setminus Y$ into a network that approximately computes the rank of i for all i.
2. Recursive application of the network obtained from the previous step, with occasional merge operations in order to correct for items that fall into the wrong recursive subproblem due to boundary effects.

If the network from Theorem 1 worked on all input permutations, and if we didn't care about constant factors, then it would be straightforward to devise an $O(\log n)$-depth sorting circuit using the approach described above. Since we do care about constant factors and since we have to worry about probabilities, however, our solution will be somewhat more involved, and the explanation will be somewhat more tedious. Nevertheless, we will still follow the basic approach described above. In the end, we will obtain a circuit with depth

$$\left(1 + \gamma + \frac{1}{1-\gamma}\right) \log n$$

that sorts a random permutation with very high probability. Using $\gamma = 0.822$ from Theorem 1, we can conclude that the sorting circuit has depth $7.44 \log n$. Although not necessarily optimal, this bound is much closer to the lower bound of $2 \log n - o(\log n)$ than previously known sorting circuits [2][7].

We begin with some definitions.

Definition 4.1 Let X denote the set of n outputs of an n-input comparator circuit C. We say that C is a *probability* p (a,b)*-closesorter* if there exists a fixed permutation π of its outputs, and a fixed subset Y of its outputs, $|Y| < 2^b$, such that on a random input permutation the probability that every output i in $X \setminus Y$ receives an input with actual rank in the open interval $(\pi(i) - 2^a, \pi(i) + 2^a)$ is at least p.

Note that a probability p $(0,0)$-closesorter must completely sort at least $pn!$ of the $n!$ possible input permutations. A probability 1 $(0,0)$-closesorter is a sorting circuit.

Definition 4.2 A *probability* p *sorting circuit* is a probability p $(0,0)$-closesorter.

Theorem 1 immediately implies the following result with $\gamma = 0.822$.

Corollary 1.1 For $n = 2^k$, the n-input butterfly comparator circuit corresponding to the butterfly tournament is a depth k, extremely high probability $(\lfloor \gamma k \rfloor + c, \lfloor \gamma k \rfloor + c)$-closesorter for some integer constant c.

The main result of this section can now be stated.

Theorem 2 Let a family of $n = 2^k$-input extremely high probability $(\lfloor \gamma k \rfloor + c, \lfloor \gamma k \rfloor + c)$-closesorters of depth k be given where γ is a real constant less than 1 and c is an integer constant. Then there exists a family of very high probability sorting circuits of depth

$$\left(1 + \gamma + \frac{1}{1-\gamma} + \epsilon\right) \log n$$

where ϵ is an arbitrarily small positive constant.

Corollary 2.1 There exists a family of very high probability sorting circuits of depth $7.44 \log n$.

Proof: Immediate from Corollary 1.1 and Theorem 2, with $\gamma = 0.822$. ☐

In the remainder of this section, the constant c refers to the constant of Corollary 1.1. Theorem 2 will be proven by using $(\lfloor \gamma k \rfloor + c, \lfloor \gamma k \rfloor + c)$-closesorters to build very high probability sorting circuits of the desired depth. In the following sequence of lemmas, it will be useful to have a compact notation for describing the degree of "sortedness" attained by a particular comparator circuit with respect to a random input permutation. We will say that a 2^k-input circuit achieves *sortedness* (k,l) or, equivalently, that it is a (k,l)*-sorter*, if it is an extremely high probability $(l,0)$-closesorter. Here one may assume that $k \geq l$, but the input size for the probability bound is defined to be 2^l, not 2^k. A (k,l)-sorter is a (k,l,s,t)-sorter if it satisfies the further condition that each of the 2^{k-s} groups of 2^s outputs sharing the same $k-s$ high order bits has been $(t,0)$-closesorted with extremely high probability; once again, the input size for the probability bound is defined to be 2^l. Square brackets will be used instead of parentheses in order to denote a deterministic level of sortedness. For instance, a true sorting circuit with 2^k inputs could be referred to as a $[k,0]$-sorter.

Lemma 4.1 A 2^k-input, probability 1 (l,l)-closesorter of depth d can be used to construct a $[k, l+2]$-sorter of depth $d + k - l - 1$.

Proof: Given such a closesorter, define X, Y and π as in Definiton 4.1, and then augment Y with arbitrary elements so that it has size 2^{l+1}. Order the outputs in $X \setminus Y$ according to the permutation π and partition them into $|Y| = 2^{l+1}$ equal-sized groups by performing an appropriate "unshuffle" operation. Note that each of these groups is sorted. Now assign one element of the set Y to each of these groups and perform a binary tree insertion. Insertion into a sorted list of length $2^r - 1$ can be performed by a simple complete binary tree circuit of depth r that uses 2^i comparators at level i, $0 \leq i < r$. In this case $r = k - l - 1$. Once all of the insertions have been performed, re-order the outputs by shuffling the resulting $|Y|$ sorted groups together. The zero-one principle can be used to check that the output is now $[k, l+2]$-sorted. Note that no assumptions have been made about the distribution of ranks in the set Y. ☐

Lemma 4.2 Assuming that $\lfloor \gamma l \rfloor + c + 2 \leq l$, a (k,l)-sorter of depth d can be used to construct a $(k, l+1, l, \lfloor \gamma l \rfloor + c + 2)$-sorter of depth $d + 2l - \lfloor \gamma l \rfloor - c - 1$.

Proof: Take the outputs of the (k,l)-sorter and perform the following steps within each of the 2^{k-l} blocks of 2^l consecutive outputs.

Apply a fixed permutation ρ, followed by a butterfly tournament comparator circuit. This requires depth l. A straightforward averaging argument, along with Corollary 1.1, shows that for the vast majority of choices of the permutation ρ, each of these blocks has been $(\lfloor\gamma l\rfloor+c, \lfloor\gamma l\rfloor+c)$-closesorted with extremely high probability. Unfortunately, the only known way to verify that a given permutation has this property requires an exponential amount of computation. Now apply Lemma 4.1. This requires depth $l-\lfloor\gamma l\rfloor-c-1$.

We need to allow for the possibility that an output that was previously within 2^l positions of its actual rank has now been moved further away by as many as $2^{\lfloor\gamma l\rfloor+c+2}$ positions. Since this quantity is assumed to be less than 2^l, every output will remain within 2^{l+1} positions of its actual rank with extremely high probability. □

Lemma 4.3 A $[k,l]$-sorter of depth d can be used to construct a $[k,0]$-sorter of depth $d+(l^2+5l+4)/2$.

Proof: Apply bitonic sort to blocks of size 2^l followed by two sets of bitonic merges between adjacent blocks. Bitonic sort requires depth $l(l+1)/2$ and each of the bitonic merges requires depth $l+1$. □

Lemma 4.4 If $k > s > l > t$, a $[k,l,s,t]$-sorter of depth d can be used to construct a $[k,t+3]$-sorter of depth $d+l-t$.

Proof: Let B_i denote the ith block of 2^s consecutive outputs, $0 \le i < 2^{k-s}$. For each i, let H_i denote the set of the highest 2^l outputs in B_i, and let L_i denote the set of the lowest 2^l outputs in B_i. Note that H_i contains every output in B_i that could possibly belong in B_{i+1}. Similarly, L_{i+1} contains every output in B_{i+1} that could belong in B_i. Only these boundary areas may need to be adjusted in order to achieve the level of sortedness required by the lemma. The condition $l < s$ guarantees that the boundary areas will not overlap. Now proceed by unshuffling each of the sets H_i and L_{i+1} into 2^{t+1} lists of size 2^{l-t-1}. Note that each of these lists is sorted. Corresponding lists are merged using a depth $l-t$ bitonic merge, and the resulting set of 2^{t+1} sorted lists of length 2^{l-t} are then shuffled together. The zero-one principle can be used to prove that the resulting outputs are indeed $[k,t+3]$-sorted. □

Now consider the following recursive procedure for constructing a very high probability sorting circuit from a (k,l)-sorter.

1. If $l \le \epsilon\sqrt{k}$ for some small positive constant ϵ, or if l is less than some appropriate positive constant, then apply Lemma 4.3 and halt. Additional depth: less than $\epsilon k + O(1)$. Sortedness: $(0,0)$-closesorter.
2. Apply Lemma 4.2 on blocks of dimension $l+2$. Additional depth: $2(l+2)-\lfloor\gamma(l+2)\rfloor-c-1$. Sortedness: $(k, l+1, l+2, \lfloor\gamma(l+2)\rfloor+c+2)$. Here we have assumed that $\lfloor\gamma(l+2)\rfloor+c+2 \le l$, which certainly holds for all values of l greater than some sufficiently large constant.
3. Apply Lemma 4.4. Additional depth: $l-\lfloor\gamma(l+2)\rfloor-c-2$. Sortedness: $(k, \lfloor\gamma(l+2)\rfloor+c+5)$.
4. Call this procedure recursively.

Let $D(k,l)$ denote the total additional depth of the circuit generated by this procedure. For $l \le \epsilon\sqrt{k}$, or when l is less than some appropriate positive constant, we have $D(k,l) \le \epsilon k + O(1)$. Otherwise,

$$D(k,l) \le (2-\gamma)l + a + D(k, \gamma l + b)$$

for some constants a and b. Solving this recurrence gives

$$D(k,l) \le \frac{(3-2\gamma)l}{1-\gamma} + O(\log l) + \epsilon k.$$

It should be emphasized that the resulting circuit is only a *very* high probability sorting circuit, even though all of the preceding lemmas hold with extremely high probability. The reason for this degradation is that the outcomes of events occurring at the leaves of the recursion are occurring with extremely high probability in terms of 2^l, which corresponds to very high probability in terms of the true input size 2^k.

Note that the total number of events that must occur in order for the sort to be successful is bounded by some polynomial in the input size 2^k. Hence, the fact that each event occurs with very high probability is sufficient to ensure that all of the events will occur with very high probability.

To obtain the best possible multiplicative constant for the leading term in the depth of our circuits, we do not apply the preceding procedure directly. Instead, we construct our circuits as follows.

1. Apply Lemma 4.2 to the entire block of 2^k inputs. Note that the empty circuit is a $[k,k]$-sorter. Depth: $2k-\lfloor\gamma k\rfloor-c-1$. Sortedness: $(k, \lfloor\gamma k\rfloor+c+2)$.
2. Apply the preceding procedure. Additional depth: $D(k, \lfloor\gamma k\rfloor+c+2)$. Sortedness: very high probability sorting circuit.

Thus, the depth of our 2^k-input, very high probability sorting circuit is

$$\left(1+\gamma+\frac{1}{1-\gamma}+\epsilon\right)k + O(\log k).$$

To obtain a proof of Theorem 2, note that the $O(\log k)$ term can be absorbed by the probability bound.

Note that there is only one step in the preceding construction for which no efficient computational procedure is known. This is the determination of an appropriate permutation π in Lemma 4.2. Fortunately, a random choice for this permutation will yield the desired performance with extremely high probability.

5 Sorting on Networks

This section sketches randomized algorithms for sorting on the hypercube and its bounded-degree variants such as as the shuffle-exchange, cube-connected cycles and butterfly. The details of these algorithms will be presented in the full paper. In the following discussion, this set of networks will be referred to as the *cube-type networks*. In the context of an n-processor fixed interconnection network, the input to the sorting problem is a set of n $O(\log n)$-bit records, distributed one per processor. The object is to determine the rank of each record and to route the record of rank i to processor i, $0 \leq i < n$.

Our strategy will be very similar to that employed in the circuit construction of Section 4. There are two main sources of additional difficulty, however. First, although the "subroutines" involved in the circuit construction are themselves amenable to cube-type computers (e.g., bitonic merge, butterfly tournament), it must be proven that the cost of permuting the data *between* subroutine calls is also $O(\log n)$. This difficulty is addressed in Section 5.1. Second, the circuit construction only provided an average case result, since randomization is not even a part of the model. In this section, our goal is to develop randomized sorting algorithms that run in $O(\log n)$ time with very high probability on *all* possible input permutations. Our randomization technique is discussed in Section 5.2.

Section 5.3 summarizes our results for sorting on cube-type networks in the word model. Section 5.4 discusses the additional details involved in obtaining a bit-serial randomized sorting algorithm for the butterfly.

5.1 A modified recursion

It is simpler to ensure that only $O(\log n)$ time is spent permuting data if we first modify the recursive sorting procedure of Section 4 in such a way that Lemmas 4.1 and 4.4 are not used. The modified construction may be sketched as follows. First, apply a butterfly tournament to the input. With extremely high probability, this brings all but $O(n^\gamma)$ of the outputs to within $O(n^\gamma)$ places of their correct output position. Second, perform a bitonic merge to bring every output to within $O(n^\gamma)$ places of its correct output position. Note that an appropriate fixed permutation must be applied before the second step. Any fixed permutation can be routed in $O(\log n)$ time on cube-type networks by precomputing the Benes paths [3]. Next, recursively sort subcubes of $O(n^\gamma)$ consecutive outputs. If these subcubes are sufficiently small, that is, if the subcube dimension is less than or equal to the square root of the original dimension, then the recursive sort is performed by applying bitonic sort. Once the recursive sort is finished, the entire sort can be completed by performing odd and even bitonic merges between adjacent sorted subcubes of $O(n^\gamma)$ outputs.

Note that the reason this particular construction was not used in Section 4 is that it leads to a multiplicative constant that is greater than 7.44.

5.2 The random butterfly tournament

The natural approach to the problem of converting a deterministic sorting algorithm that sorts a randomly chosen input permutation in $O(\log n)$ time with (very, extremely) high probability into a randomized algorithm that sorts every input permutation in $O(\log n)$ time with (very, extremely) high probability, is to look for a randomized algorithm to route a random permutation in $O(\log n)$ time with (very, extremely) high probability. The approach most often used for generating a random permutation is based upon the idea of sending each input to a random destination. For technical reasons, this method seems to be limited to a "high" probability of success, and is not suitable for use with $O(\log n)$-bit step algorithms.

Definition 5.1 A *random butterfly tournament* is a butterfly tournament in which the outcome of each match is determined by the toss of a fair coin.

Even though the output of a random butterfly tournament is *not* a random permutation of the input, we will prove that it is sufficiently random to allow the preceding reduction to go through. Furthermore, the resulting randomized sorting algorithms will run in $O(\log n)$ time with *very* high probability.

Lemma 5.1 Every participant in a random butterfly tournament with $n = 2^d$ players is equally likely to achieve any particular W-L sequence.

Proof: Straightforward. □

In the following sequence of lemmas, assume that the input to a random butterfly tournament consists of k 0's and $n-k$ 1's, where $n = 2^d$. Let $p = k/n$. Also, at depth d' of the randomization pass, we partition the outputs into $2^{d'}$ "intermediate groups" of $2^{d-d'}$ consecutive outputs.

Lemma 5.2 Let X_i denote the random variable equal to the number of 0's in the ith intermediate group. Suppose that $d' = \Theta(d)$, and let ϵ denote an arbitrarily small positive constant. Then

$$\max_{0 \le i < 2^{d-d'}} |X_i - p2^{d-d'}| = O(2^{(d-d')/2+\epsilon})$$

with extremely high probability.

Proof: The first d' levels of the butterfly can be partitioned into $2^{d-d'}$ butterflies of order d'. Each of these butterflies contributes a single output to each of the intermediate groups. Hence, Lemma 5.1 implies that the random variable X_i is an (unweighted) sum of $2^{d-d'}$ independent Bernoulli trials, where the probability of success in the jth trial, p_j, is given by the fraction of 0 inputs to the jth butterfly, $0 \le j < 2^{d-d'}$. The expected number of 0's in each intermediate group is $\sum_{0 \le j < 2^{d-d'}} p_j = p2^{d-d'}$.

Standard Chernoff-type bounds can now be applied to obtain the stated inequality [4][8]. □

Using Lemma 5.1, the last $d - d'$ stages of the random butterfly tournament may be interpreted as re-partitioning the input into $2^{d'}$ "output groups" of size $2^{d-d'}$, where the ith output group receives exactly one element uniformly at random from each of the intermediate groups. Thus, we have the following corollary to Lemma 5.2.

Corollary 2.2 Suppose that $d' = \Theta(d)$, and let ϵ denote an arbitrarily small positive constant. Given the output values for any subset of the rest of its output group, the conditional probability p' of any particular output being a 0 satisfies

$$p - O(2^{\epsilon - d'/2}) \le p' \le p + O(2^{\epsilon - d'/2})$$

with extremely high probability.

Proof: This condition is satisfied exactly when the condition of Lemma 5.2 is satisfied. □

Thus, there is a limited amount of dependence between the 0–1 assignments within each output group. By making use of a certain monotone property of comparator circuits, we are able to prove that the butterfly tournament remains an effective sorting subroutine when there is a similarly limited amount of dependence between the assignments to its inputs.

The preceding results lead to the following subroutine for improving the sortedness of an arbitrary input permutation of size n with extremely high probability. Assume that d' and $d - d'$ are both $\Theta(d)$.

1. Perform a random butterfly tournament over the entire set of n records.
2. Partition the n records into $2^{d'}$ output groups of size $m = 2^{d-d'}$, as described above.
3. Run a butterfly tournament over each of the output groups in parallel. As argued above, Theorem 1 can be applied to each of the groups individually. Let X_i, Y_i denote the sets X, Y of Theorem 1 corresponding to group i. Let $Z_i = X_i \setminus Y_i$.
4. Shuffle together the Z_i sets into a set A and concatenate the Y_i sets to obtain a set B. Note that A and B partition the original set of n records. A simple calculation shows that $|B| = O(n^{-\epsilon})$ for some positive constant ϵ. Using the zero-one principle and Lemma 5.2, we can show that every record in A is within $n^{1-\epsilon}$ positions of its correct location with extremely high probability.

Thus, we can exploit the limited randomness provided by a random butterfly tournament in order to improve the sortedness of the input by a polynomial factor with extremely high probability.

5.3 Sorting in the word model

In the word model it is assumed that the processors of an n-node fixed interconnection network can execute instructions on $O(\log n)$-bit operands in constant time. The cost of sending an $O(\log n)$-bit message to an adjacent processor is also assumed to be constant. We have obtained the following result.

Theorem 3 There exist randomized sorting algorithms that run in $O(\log n)$ time with very high probability on cube-type computers under the word model.

Furthermore, these algorithms are completely constructive.

5.4 Bit-serial sorting on the butterfly

This section describes a bit-serial sorting algorithm for sorting on the butterfly. In the bit model, it is assumed that a processor can only perform one bit operation per time step. Thus, m time steps are required to send an m-bit message to an adjacent processor. In the bit model, one cannot hope to sort n $O(\log n)$-bit records on an n-node bounded degree network in $O(\log n)$ bit steps; to the contrary, there is a trivial $\Omega(\log^2 n)$ lower bound. In order to achieve $O(\log n)$ bit steps one must consider sorting on a network that is lightly loaded by a $\log n$ factor. Thus, the following result is asymptotically optimal.

Theorem 4 There is a randomized algorithm for sorting n $O(m)$ bit records on an $n \log n$ node butterfly network that runs in $O(m + \log n)$ bit steps with very high probability.

Given the techniques discussed earlier in Section 5, only one difficulty remains to be overcome in order to prove this theorem. A naive implementation of the sorting algorithm described in Section 5.1 would make use of fewer and fewer of the $\log n$ rows of butterfly nodes to realize the subsorts occurring at deeper and deeper levels of the recursion, and certain butterfly nodes (those near the outputs) would participate in every level of the recursion. Since there are $O(\log \log n)$ levels of recursion, and a node performs $O(m)$ bit operations with respect to each level of the recursion in which it participates, such an algorithm leads to an $O(m \log \log n + \log n)$ bound. It turns out that this can be reduced to the desired $O(m + \log n)$ bound by organizing the flow of data in such a way that each row of butterfly nodes participates in only a constant number of levels of the recursion. The details of this implementation will be presented in the full paper.

6 Concluding Remarks

While the multiplicative constant of 7.44 proven for the sorting circuit construction of Section 4 appears to be quite reasonable, the construction remains impractical. This is due to the fact that there is a tradeoff between the value of the multiplicative constant and the success probability (the probability that a random input permutation is sorted by the circuit), and for practical values of n, a significant increase in the constant is required in order to prove any reasonable success probability.

On the other hand, there appear to be a number of possible avenues to explore in terms of making the construction more practical, and our research in this direction is ongoing. In particular, we have recently implemented a circuit construction algorithm that employs heuristics based upon the theory developed in this paper, and the preliminary results are quite encouraging. For example, simulations performed by Yuan Ma indicate that we can construct a probability 0.99 1024-input sorting circuit with much smaller depth than the standard 1024-input bitonic sorting circuit. In addition, our heuristic circuits appear to possess a significant degree of fault tolerance. The details of the experimental work will appear in a subsequent paper.

7 Acknowledgments

Thanks to Don Coppersmith, Ming Kao, Yuan Ma, and Bruce Maggs for stimulating discussions.

References

[1] B. Aiello, F. T. Leighton, B. Maggs, and M. Newman. Fast algorithms for bit-serial routing on a hypercube. In *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 55–64, 1990.

[2] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. *Combinatorica*, 3:1–19, 1983.

[3] V. E. Benes. Optimal rearrangeable multistage connecting networks. *Bell System Technical Journal*, 43:1641–1656, 1964.

[4] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–509, 1952.

[5] D. E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, Reading, MA, 1973.

[6] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. Unpublished manuscript, October 1989.

[7] M. S. Paterson. Improved sorting networks with $O(\log n)$ depth. *Algorithmica*, 5:75–92, 1990.

[8] P. Raghavan. Probabilistic construction of deterministic algorithms approximating packing integer programs. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 10–18, 1986.

[9] J. H. Reif and L. G. Valiant. A logarithmic time sort for linear size networks. *JACM*, 34:60–76, 1987.