Highly Fault-Tolerant Sorting Circuits

Tom Leighton
M.I.T.
Cambridge, MA 02139

Yuan Ma
M.I.T.
Cambridge, MA 02139

C. Greg Plaxton
University of Texas
Austin, TX 78712

Abstract

In this paper, we consider the problem of constructing a sorting circuit that will work well even if a constant fraction of its comparators fail at random. We consider two types of comparator failures: *passive* failures, which result in no comparison being made (i.e., the items being compared are output in the same order that they are input), and *destructive* failures, which result in the items being output in the reverse of the correct order. In either scenario, we assume that each comparator is faulty with some constant probability ρ, and we say that a circuit is fault-tolerant if it performs some desired function with high probability given that each comparator fails with probability ρ. Our results include:

(i) the construction of a passive-fault-tolerant circuit of depth $O(\log n \log\log n)$ that sorts most permutations of n inputs,

(ii) a proof that any destructive-fault-tolerant circuit that is capable of approximate insertion (or many other operations) must have depth $\Omega(\log^2 n)$,

(iii) the construction of a destructive-fault-tolerant circuit of depth $O(\log^2 n)$ that approximately merges all approximately sorted lists of size n, and

(iv) the construction of a destructive-fault-tolerant circuit of depth $O(\log^2 n)$ that approximately sorts most permutations of n inputs.

1 Introduction

Although sorting circuits have been extensively studied [1, 4, 8, 12], relatively little is known about the behavior of sorting circuits when comparators are allowed to be faulty. Past work on fault-tolerant sorting circuits [3, 14, 15] has focused on two types of comparator failures. The simplest type of failure is a *passive* failure, in which the faulty comparator does nothing (i.e., the items being compared are output in the same order that they are input). Yao and Yao [15] studied comparator circuits in this model, and proved tight bounds for the problem of finding the minimum of n items when a constant fraction of the components experience passive failure. They do not study the problem of sorting in this model, other than to point out that any sorting circuit can be made tolerant to random passive faults by replicating each comparator $O(\log n)$ times (since it is likely that one of the copies is working and so the functionality of the circuit is preserved). This results in a passive-fault-tolerant sorting circuit with depth $O(\log^2 n)$. Yao and Yao leave open the question of whether or not there is a passive-fault-tolerant circuit with $o(\log^2 n)$ depth that sorts all input permutations with high probability. We partially resolve this question by exhibiting a circuit with $O(\log n \log\log n)$ depth that sorts any fixed permutation with high probability even if each comparator suffers a passive failure with probability $\rho < 1$.

In [3], Assaf and Upfal consider a *destructive* model of comparator failure in which the items being compared are output in the wrong order. (Note that a passively faulty comparator can output items in the wrong order, but only if the items are input in the wrong order.) Destructive comparator failures are particularly troublesome because they can "unsort" items that have been previously put into sorted order. Indeed, no circuit that suffers destructive failure in a constant fraction of its comparators is likely to be able to completely sort any fixed permutation of the inputs. This is because (as Assaf and Upfal note) there are many comparators in the last level that are likely to be faulty, and hence, many items are likely to be out of order after passing through the circuit. In fact, it is possible to employ a more complicated analysis to show that at least one of the n items is likely to be at least $\Omega(\log n)$ positions out of order after passing through any circuit that suffers destructive comparator failure with constant probability.

Faced with such difficulties, there are two possible courses of action: we can relax the usual constraints associated with the comparator circuit model, or we can stick to the classical model and try to design circuits that "almost" sort. Assaf and Upfal pursued the former approach, and showed that by allowing fan-out 2 in the comparator circuit model (i.e., allowing the output of a particular gate to be connected to two inputs at a later stage), any sorting circuit can be made tolerant to random destructive failures at a cost of a $\Theta(\log n)$-factor in hardware. Interestingly, this $\Theta(\log n)$ blow-up does not apply to the depth of the

This research was supported by 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.

circuit, but only to its width (the circuit has n input registers and n output registers, but use of fan-out results in a constant fraction of the intermediate levels having $\Theta(n \log n)$ registers).

In this paper, we pursue the latter approach. We start by proving that in order for a comparator circuit to do anything reasonable (such as inserting an item into a sorted list without damaging the sortedness significantly) in the presence of random destructive faults, the circuit must have $\Omega(\log^2 n)$ depth. We then prove several positive results. Most importantly, we construct a circuit with $O(\log^3 n)$ depth that with high probability is capable of approximately sorting any n inputs even if a constant fraction of the comparators suffer destructive faults. (In this case, the probability depends only on the randomness of the faults, i.e., the circuit is likely to work for *all* input permutations). By *approximately sorting* n items, we mean that every item is sorted to within $\log n$ distance of its correct position in the total order, which is optimal for circuits with destructive failures. The result is based on an interesting analysis of odd-even transposition sort in which comparators can suffer random destructive failures. We also describe an $O(\log^2 n)$-depth circuit that will be able to approximately sort most (but not necessarily all) input permutations even if a constant fraction of its comparators suffer random destructive failures. As a consequence, we find that the impact of destructive failures on sorting circuits is not so severe after all. Items can still be sorted to within $\log n$ distance of their correct destinations, although greater depth is required to do this than if the faults were passive.

We also study the closely related problem of tournament ranking in the presence of uncertainty. In particular, we consider the problem of designing and ranking a $\log n$-round, n-player tournament where there is an underlying (but completely unknown) total order on the players, and where every match results in an upset (i.e., where the weaker player beats the stronger player) with some fixed (and known) probability. Somewhat surprisingly, we show that there is a ranking for the butterfly tournament such that with high probability, the rank of all but n^γ players can be correctly determined to within n^γ positions, where γ is a positive constant, $\gamma < 1$. In fact, we show how to compute the ranking without knowing which matches resulted in upsets. The ranking is sensitive to knowing the probability of upsets however. As a consequence of our work on sorting circuits, we can also design a tournament where the rank of every player is correctly computed to within $\log n$ positions with high probability, and where the rankings are not dependent on knowing the precise probability of an upset (only an upper bound on the upset probability is needed), but this tournament has $O(\log^2 n)$ rounds.

The remainder of the paper is divided into sections as follows. Section 2 contains some definitions and terminology that will be used throughout the paper. Section 3 describes the general framework for several of our sorting circuit constructions. Section 4 contains our results on $\log n$-round tournament ranking in the face of uncertainty. Section 5 contains our results on sorting circuit with passive faults. Section 6 contains our results on comparator circuits with destructive faults. Due to space limitations, we have only included sketches of most of the proofs.

2 Notation and Definitions

In the fault-free model, a comparator gate is a 2-input, 2-output device with one output labelled MAX and the other output labelled MIN. The inputs are not labelled. On input x and y the comparator gate routes $\max\{x, y\}$ to the MAX output and $\min\{x, y\}$ to the MIN output. In our idealized fault-tolerant circuits, the only kind of fault that can occur is that a comparator gate may produce an "unsorted" output, in other words, it may interchange the two output values. Note that we do not consider more malicious kinds of faults that might lead to corruption and/or loss of data. In order to distinguish between our two basic fault models, it is useful to augment the above definition of a comparator gate by assuming that the two inputs are also labelled MAX and MIN. Having done so, we can now identify two distinct kinds of faults. Under the first kind of fault, which we refer to as a *passive* fault, the input arrives "unsorted" (i.e., $\max\{x, y\}$ arrives at the MIN input and $\min\{x, y\}$ arrive at the MAX input) and leaves "unsorted". Under the second kind of fault, which we refer to as a *destructive* fault, the input arrives "sorted" and leaves "unsorted". Intuitively, we expect that faults of the latter kind will be the most difficult to cope with, which explains our choice of nomenclature.

We will often be interested in circuits that perform some prescribed task *with high probability*. This phrase implies that the failure probability of the circuit can be made less than n^{-c} for any positive constant c, where n is a fixed parameter that may be unrelated to the number of inputs of the circuit.

2.1 Models of Computation

In this paper, we study the depth complexity of fault-tolerant comparator circuits for performing a variety of sorting-related tasks (e.g., sorting, merging, insertion). We consider two basic fault models as well as a number of different complexity measures. In order to make the statements of our results as precise as possible, each will be described in terms of a number of parameters. For example, we will see that the AKS $O(\log n)$-depth sorting circuit result would be expressed in our notation as

$$|\mathtt{Sort}(n, 0, 0, (Z, I, 0, 0))| = O(\log n).$$

Note that the `Sort` problem has 4 arguments, the last of which is itself a 4-tuple. The last argument of every problem that we consider will be such a 4-tuple, and specifies the *model of computation* of the result. In general, a model of computation takes the form $(\sigma, \tau, \rho, \epsilon)$ where σ belongs to $\{D, P, Z\}$, τ belongs to $\{I, II, III, IV\}$, and ρ, ϵ are probabilities.

We now define the model of computation in terms of σ, τ, ρ, and ϵ. The values of σ and ρ determine the nature of the faults that can occur in the comparator circuit. In particular, the ith gate of the comparator circuit will be independently set to "faulty" with some probability $\rho_i \leq \rho$. The meaning of the term "faulty" depends on the value of the parameter σ.

If $\sigma = Z$ (the zero-fault or fault-free model) then ρ must be equal to 0 and every gate in the circuit is guaranteed to function correctly. In other words, $\sigma = Z$ denotes the standard comparator circuit model.

If $\sigma = P$ (the passive fault model), a faulty gate only commits passive faults. Note that labelling a gate as faulty is then equivalent to removing the gate from the circuit by connecting the MAX input of the gate to the MAX output and the MIN input to the MIN output. Under this fault model, we will generally assume that the fault probability ρ is bounded away from 1. Given this assumption, we can simulate an arbitrarily small constant fault probability by sequentially repeating each comparison a constant number of times. We will make extensive use of this "redundancy" trick; in fact, we will have occasion to apply non-constant redundancy in order to achieve a vanishingly small fault probability. As indicated in the introduction, $O(\log n)$ redundancy is sufficient to ensure that every comparison in a polynomial-size circuit will be performed correctly with high probability.

If $\sigma = D$ (the destructive fault model), a faulty gate commits both passive and destructive faults. In other words, the output of a faulty gate is always unsorted. In this fault model, we require that the fault probability ρ be bounded below $1/2$ (a comparator gate that is faulty with probability $1/2$ is of no use, and ρ is only an upper bound on the fault probability). As in the fault-free model, there is no significance to the labelling of the inputs to a comparator gate. It is worth mentioning that the redundancy trick employed in the passive fault model is of no use here, since repeating the same comparison $k \geq 1$ times will yield a sorted result if and only if the last comparator gate in the sequence is non-faulty. As a consequence, it is not clear that an upper bound proven for $\rho = 1/4$, say, can be extended to an upper bound for $\rho = 1/3$. In fact, all of our upper bound results in the destructive model require that ρ be at most some small positive constant.

We will now complete our definition of the model of computation by explaining the role of the parameters τ and ϵ. Assume that a given comparator circuit performs some sorting-related task under the model of computation $(\sigma, \tau, \rho, \epsilon)$. The parameter τ denotes the *type* of the circuit. A circuit is of Type *I* if and only if it works correctly on all possible inputs with probability at least $1-\epsilon$. A circuit is of Type *II* if and only if for any fixed input it works correctly with probability at least $1-\epsilon$. A circuit is of Type *III* if and only if it works correctly on a randomly chosen input (i.e., chosen uniformly over the space of possible inputs) with probability at least $1-\epsilon$. A circuit is of Type *IV* if and only if it satisfies the Type *III* requirements whenever the failure probability of each comparator gate is *exactly* ρ (rather than being less than or equal to ρ). Note that there is a strict hierarchy amongst these four circuit types: the Type *I* circuits form a proper subset of the Type *II* circuits, the Type *II* circuits form a proper subset of the Type *III* circuits, and the Type *III* circuits form a proper subset of the Type *IV* circuits. Thus, with respect to these four types of circuits, the strongest upper bounds are of Type *I* and the strongest lower bounds are of Type *IV*. One of our upper bounds is of Type *IV*, and the rest are either of Type *I* or of Type *II*. Our lower bounds are all of Type *IV*.

2.2 Sorting-Related Problems

While the main focus of this paper is on the depth complexity of fault-tolerant sorting circuits, our constructions will make use of a number of "subroutines" for solving other sorting-related problems. In particular, we will have occasion to devise circuits for approximate sorting, insertion, merging, and extraction. Furthermore, our main lower bound deals with the complexity of a particular concatenation problem. All of these sorting-related problems are formally defined below. In the following, let $\mathcal{M} = (\sigma, \tau, \rho, \epsilon)$ denote an arbitrary model of computation, and define a list to be Δ-*approximately sorted* if and only if every item is sorted to within distance Δ of its correct position.

Sorting. We define a 4-parameter sorting problem, $\texttt{Sort}(n, \Delta, \Gamma, \mathcal{M})$. The problem is to produce a Δ-approximately sorted list of length at least $n - \Gamma$. Note that in the special case $\Delta = \Gamma = 0$ this corresponds to the usual sorting problem.

Insertion. Given a sorted list of length $n-1$ and an additional item, the problem $\texttt{Insert}(n, \Delta, \mathcal{M})$ is to produce a Δ-approximately sorted list of length n.

Merging. Given two Δ-approximately sorted lists of length m and n, the problem $\texttt{Merge}(m, n, \Delta, \mathcal{M})$ is to produce a single Δ-approximately sorted list of length $m+n$.

Extraction. The problem $\texttt{Extract}(n, k, \Delta, \mathcal{M})$ is to move the k smallest (largest) of n given inputs to a particular set of Δ outputs.

Maximum. $\texttt{Maximum}(n, \Delta, \mathcal{M})$ denotes the special case $\texttt{Extract}(n, 1, \Delta, \mathcal{M})$.

Concatenation. $\texttt{Concatenate}(n, \Delta, \mathcal{M})$ denotes a special case of the insertion problem in which the item to be inserted is known to be either the smallest or the largest item. In other words, the input is either $(0, 1, \ldots, n-1)$ or $(1, 2, \ldots, n-1, 0)$, and the output is a Δ-approximately sorted list of length n.

In Section 6.1 we will prove that even for large values of Δ, the `Maximum` and `Concatenate` problems are not easy to solve in the destructive model.

Given a problem P, we let $|P|$ denote the minimum depth circuit required to solve P.

3 A Framework for Efficient Sorting

In Section 5.3 we will describe a circuit construction that establishes

$$|\mathtt{Sort}(n, 0, 0, (P, II, \rho, n^{-c}))| = O(\log n \log\log n) \qquad (1)$$

for any positive constant c, and for any constant fault probability $\rho < 1$. This construction makes extensive use of a modified version of the AKS sorting circuit. As a result, the constant hidden by the O-notation is rather large. A Type IV construction yielding the same asymptotic bound but having a more modest multiplicative constant is given in Section 5.2. The modified AKS approach is exploited once again in Section 6.5 to prove that

$$|\mathtt{Sort}(n, \Theta(\log n), 0, (D, II, \rho, n^{-c}))| = O(\log^2 n) \qquad (2)$$

for some positive constant c, and for any sufficiently small positive constant ρ. While each of these constructions is based upon a markedly different design for certain specific components, they all adhere to the same high-level organization. In particular, they all make use of a simple technique for reducing a given sorting problem to a number of smaller and/or coarser approximate sorting and insertion problems. In this framework, we view the sorting problem as a limiting case of approximate sorting (as we have in the expressions above). Let

$$\begin{aligned}
\mathcal{M} &= (\sigma, \tau, \rho, n^{-c}),\\
S_0(m) &= |\mathtt{Sort}(m, 0, 0, \mathcal{M})|,\\
S_1(m,\gamma) &= |\mathtt{Sort}(m, \Theta(m^\gamma), 0, \mathcal{M})|,\\
S_2(m,\gamma) &= |\mathtt{Sort}(m, \Theta(m^\gamma), \Theta(m^\gamma), \mathcal{M})|, \text{ and}\\
I(m,\gamma) &= |\mathtt{Insert}(m, \Theta(m^\gamma), \mathcal{M})|.
\end{aligned}$$

Once a list has been Δ-approximately sorted, it can be completed sorted by performing two sets of parallel sorts on contiguous blocks of size 2Δ (the second set of sorts is offset from the first set by distance Δ). Hence, we have

$$S_0(n) \le S_1(n,\xi) + 2S_0(\Theta(n^\xi)) \qquad (3)$$

for any $\xi < 1$. In our constructions, we will set ξ to either $1/\sqrt{\log n}$ or $\Theta(1/\log n)$. We choose such a cut-off by determining the level at which we can give an effective direct bound (i.e., without recursion) for the $S_0(\Theta(n^\xi))$ terms. Such a direct bound might be obtained by making use of a fault-tolerant version of bitonic sort [4], for example. Thus, our main challenge is to provide a bound for $S_1(n,\xi)$.

To bound $S_1(n,\xi)$, we solve a given approximate sorting problem in terms of: (i) a single, coarser approximation problem with the same number of inputs, and (ii) two disjoint sets of smaller approximate sorting and insertion problems (each set can be performed in parallel). This technique leads to the recurrence

$$\begin{aligned}
S_1(n,\alpha') \quad &\le \quad S_1(n,\alpha) + 2S_2(\Theta(n^\alpha), \beta) +\\
&\qquad 2I(\Theta(n^{\alpha(1-\beta)}), \gamma),
\end{aligned}$$

where $\alpha' = \alpha(\beta + \gamma - \beta\gamma)$, and α, β, γ belong to $(0,1)$. We apply this recurrence by exhibiting circuits that provide upper bounds on $S_2(m,\beta)$ and $I(m,\gamma)$ for all $m \le n$ and specific constants β, γ less than 1. Note that $\delta \stackrel{\text{def}}{=} \beta + \gamma - \beta\gamma < 1$, so that $\alpha' < \alpha$. Thus the recurrence leads to a bound of essentially (we are ignoring some inconsequential effects due to the appearances of Θ above)

$$S_1(n,\xi) \le 2 \sum_{0 \le i \le s} \left[S_2(n^{\delta^i}, \beta) + I(n^{\delta^i(1-\beta)}, \gamma) \right], \qquad (4)$$

where $s = \lceil \log_\delta \xi \rceil$. The reader is referred to [10, 9] for a more detailed account of a similar recursive sorting strategy. In the fault-free model, the scheme described in [10] is preferred because it leads to a smaller constant factor. For investigating asymptotic fault-tolerant sorting bounds, however, the scheme presented above has the advantage that it requires only approximate sorting and insertion routines; it does not require a merging routine. In the passive model, we do not know how to perform merges as efficiently as insertions, and so this advantage is critical. In the destructive fault model this is not the case, and so we could just as easily make use of the recursion described in [10].

4 Tournament Results

In [10], the butterfy tournament was analyzed and employed to obtain small-depth sorting circuits that sort the vast majority of all $n!$ randomly chosen permutations. In fact, for a randomly chosen input permutation, the probability of failure of these circuits is asymptotically smaller than $1/n^c$ for any constant c.

In Section 4.1, we present a fault-tolerant extension of the main theorem presented in [10], which demonstrated that the outputs of an n-input butterfly tournament can be labelled with a permutation of $[0, n-1]$ in such a way that a surprisingly strong ranking property is revealed. An unfortunate shortcoming of our fault-tolerant result is an inherent lack of robustness, an issue that is discussed in Section 4.2.

4.1 Tournament Analysis

In the following discussion, assume that $m = 2^k$, $m \le n$. An m-input butterfly tournament of order k may be defined as follows. In the first round, $m/2$ matches are played according to a random pairing of the m players. The next $k-1$ rounds are defined by recursively running a tournament amongst the $m/2$ winners, and (in parallel) a separate tournament amongst the $m/2$ losers. Note that an m-input butterfly tournament corresponds in a natural way to an m-input comparator circuit based on the butterfly organization. The following theorem was proven in [10].

Theorem 1 Let $X = \{0, \ldots, m-1\}$. Then there exists a fixed permutation π of X, a positive constant γ strictly less than 1, and a fixed subset Y of X such that $|Y| = O(m^\gamma)$, and the following statement holds with probability at least $1 - 2^{-m^\epsilon}$ (ϵ constant, $\epsilon > 0$): If m players participate in a butterfly tournament, then the actual rank of player i lies in the range $[\pi(i) - O(m^\gamma), \pi(i) + O(m^\gamma)]$ for all i in $X \setminus Y$.

We have obtained the following similar result for the case where each gate in the m-input butterfly comparator circuit is independently faulty with probability *exactly* ρ. In the passive fault model, we require that ρ be bounded below 1. In the destructive fault model, we require that ρ be bounded away from 1/2.

Theorem 2 For a ρ-faulty butterfly tournament, the statement of Theorem 1 holds with the bound $1 - 2^{-m^\epsilon}$ weakened to $1 - m^{-c}$ (where c denotes an arbitrary positive constant).

While the fault-tolerant theorem bears a strong similarity to its fault-free counterpart, several new ideas were needed in order to carry out the proof. Unfortunately, an adequate presentation of these ideas would require a substantial review of the analysis presented in [10]. In view of space limitations, we have decided to postpone this material to the full version of the paper.

In the passive fault model, it turns out that we can improve the probability of failure to "high probability" (i.e., polynomially small in terms of n) if $\rho \leq c^{\log_m n}$ for some positive constant c, $c < 1$. Since we can simulate such a value of ρ by repeating each comparison $O(\log_m n)$ times, we can obtain an $O(\log n)$-depth circuit that satisfies the criterion of Theorem 2 with high probability in terms of n.

4.2 Robustness

It is interesting to note that the tournament result of the preceding section is not robust under either the passive or the destructive fault model. In other words, we require a predictable failure rate ρ for our comparator gates; it is not enough for ρ to represent an upper bound on the actual failure rate. To see why this is the case, consider what happens if we apply a randomly chosen 0-1 sequence to an n-input butterfly tournament in which each gate is faulty (we will assume the destructive model; a similar analysis applies to the passive model) with probability exactly ρ. With high probability, the initial sequence will contain $n/2 \pm O(\sqrt{n})$ 0's (and a similar number of 1's). The first set of $n/2$ comparisons partition the n inputs into two sets of size $n/2$, the "winners" (call this set W) and the "losers" (call this set L). Thereafter, no comparison is performed between a member of the set W and a member of the set L. Furthermore, the set W contains $n(1+2\rho)/8 \pm O(\sqrt{n})$ 0's and $n(3-2\rho)/8 \pm O(\sqrt{n})$ 1's with high probability (and vice versa for the set L).

Now the permutation π discussed in the preceding section must map a fixed number, say k, of the items in the set W to the high half of the final ranking (of all n output positions), and $n/2 - k$ to the low half. Hence, the number of outputs in W that will be assigned to the wrong half will be

$$w = \Omega(|k - n(3-2\rho)/8|)$$

with high probability. Now suppose that ρ is only an upper bound on the failure rate of our gates and the actual failure rate is some ρ', $\rho' < \rho$. In that case, the number of outputs assigned to the wrong half will be $w' = \Omega(|k - n(3-2\rho')/8|)$. Note that $w + w' = \Omega(n(\rho - \rho'))$. In particular, if $\rho' = 0$ then $w + w' = \Omega(n\rho)$. Furthermore, $w + w' = \Theta(\max\{w, w'\})$, and so the permutation π must err by $\Omega(n\rho)$ either when none of the gates are faulty or when each gate is faulty with probability ρ.

To summarize, the lack of robustness in Theorem 2 does not reflect a gap in our analysis; rather, it is indicative of an inherent shortcoming in the butterfly tournament sorting technique.

5 Results for the Passive Model

This section describes our results for the passive fault model. Section 5.1 describes an $O(\log n)$-depth circuit for insertion. Section 5.2 indicates how to combine this circuit with the butterfly tournament primitive of Section 4.1 in order to obtain a Type *IV* $O(\log n \log\log n)$-depth sorting circuit. Section 5.3 sketches the construction of a Type *II* $O(\log n \log\log n)$-depth sorting circuit based on the techniques of AKS. Throughout this section, let

$$\begin{aligned} \mathcal{A} &= (P, I, \rho, n^{-c}) \text{ and} \\ \mathcal{B} &= (P, II, \rho, n^{-c}), \end{aligned}$$

where $\rho < 1$ and c denotes an arbitrary positive constant.

5.1 Insertion

A number of authors have studied the problem of binary search in a faulty environment (see, for example, [5, 7, 13]). However, none of these results seem to extend to the faulty comparator circuit model. The main result of this section is summarized by the following theorem.

Theorem 3 For all $m \leq n$ and some constant γ, $0 < \gamma < 1$, we have

$$|\texttt{Insert}(m, m^\gamma, \mathcal{A})| = O(\log n).$$

We prove the preceding theorem by exhibiting an appropriate family of circuits. In the following, assume that $m = 2^l$, where l is a positive integer. The input to the circuit consists of a sorted list L of length $m-1$ and an "unknown" item α.

Rather than solving the given insertion problem "directly", we first partition the $m-1$ items of the sorted list L into $r = 2^{\lfloor l/5 \rfloor}$ contiguous "pools" of size $s = 2^{l - \lfloor l/5 \rfloor}$ (except for the last pool, which has only $s-1$ items). Conceptually, we now solve an insertion problem with only $r+1$ inputs (one for each pool, plus the unknown input α), and attempt to determine approximately which pool the input α belongs to. It is useful to think of all of the items in any one pool as being indistinguishable.

Our construction will have the property that the MAX input of every comparator gate is always a "pool input" (i.e., an item from the list L that has never been compared against anything else). Furthermore, we will never compare two pool inputs directly. Under these restrictions, it is easy to see that our comparator circuit will correspond to a binary tree. At the first level of the circuit, there is a single comparator gate. The unknown input α is inserted in the MIN input of that gate, and an input from some pool j is inserted in the MAX input, $0 \le j < r$. Thus, all that remains to be specified by the circuit construction (with respect to the first level of the circuit) is the value of j. The second level of the circuit will have two comparator gates. The MIN output of the gate from the first level is fed into the MIN input of one of the two gates, and the MAX output is fed into the other MIN input. The circuit construction must specify from which (possibly distinct) pools the two MAX inputs will be drawn.

Our circuit will correspond to a complete binary tree of depth $d-1$ (the root is at depth 0, and the leaves are at depth $d-1$). In other words, each output of a gate at level i will be fed into the MIN input of some gate at level $i+1$, $0 \le i < d-1$. A circuit of this type is defined by stating, for each string of MINs and MAXs of length less than or equal to d, which pool the MAX input will be taken from. Note that there are a total of $2^d - 1$ MAX inputs to be specified. Thus, we cannot possibly exhaust any particular pool (the smallest being of size $s-1$) if $2^d \le s$. It will turn out that our circuit has depth $d = 4 \log r$, implying that the preceding inequality is indeed satisfied.

We now describe a procedure for computing which pool the MAX input of each gate in the circuit is to be taken from. As suggested above, there is an obvious 1-1 correspondence between the wires of our circuit and the set of strings of length less than or equal to d over the two-symbol alphabet MIN and MAX. The binary strings of length exactly d correspond to outputs of the circuit and there is no need for the algorithm to specify a corresponding MAX input. For each binary string σ of length i, $0 \le i < d$, the algorithm executes the following procedure in order to specify the pool number of the correponding MAX input.

1. Initialize an infinite, complete binary tree with nodes labelled as follows. The r nodes at depth $\log r = \lfloor l/5 \rfloor$ are labelled 0 through $r-1$ (in order). Every node at depth greater than $\log r$ is labelled with the label of its ancestor at level $\log r$. Every node at depth less than $\log r$ is labelled with the least label in its right subtree.
2. Place a token at the root of the tree.
3. Repeatedly update the tree (and the position of the token within the tree) according to the sequence of MINs and MAXs given by the string σ (in the order specified by σ), and the operations defined below.
   (a) **MIN update.** Move the token to the left child of the node that it currently resides in.
   (b) **MAX update.** If the token is at the root, then move the token to the right child of the root, delete the root, and delete the left subtree of the root (i.e., the right child of the root becomes the new root). If the token is at some node x which is not the root, then move the token to the parent of x, delete x, delete the left subtree of x, and "raise" the right subtree of x by making the right child of x the left child of the parent of x (i.e., the right child of x replaces x in that role).
4. Return the label of the node at which the token is located.

Our procedure for constructing insertion circuits has now been specified, except for one minor detail. For $m \ll n$, the preceding construction does not lead to a high probability bound. To rectify this, we apply the redundancy trick described in Section 2.1. In particular, we will repeat each comparison $\Theta(\log_m n)$ times. Note that such redundant comparisons do not make use of additional pool inputs, and will have no impact on the analysis except to increase the overall depth (and size) of the circuit by a factor of $\Theta(\log_m n)$.

Due to space limitations, we have omitted the analysis of the preceding construction. We claim that a circuit satisfying the bound of Theorem 3 can be obtained by applying our construction to depth $4 \log r$ with $\Theta(\log_m n)$ redundancy.

5.2 A Type IV Sorting Circuit

Given the passive model result of Section 4.1 and the insertion result of Section 5.1, it is now a relatively simple task to prove a Type IV version of Equation 1. We prove this claim by constructing a circuit within the framework described in Section 3. We will set ξ to $\Theta(1/\log n)$ in Equation 3. It is immediate that $S_0(\Theta(n^\xi)) = O(\log n)$ because we can afford to apply bitonic sort with $O(\log n)$ redundancy to subproblems of constant size.

Let us now turn our attention to the RHS of Equation 4. By Theorem 3, each of the insertion terms is equal to $O(\log n)$. Furthermore, the butterfly tournament result of Section 4.1 implies that each approximate sorting term is also $O(\log n)$. Hence, Equations 3 and 4 give $S_0(n) = S_1(n, \xi) = O(\log n \log \log n)$.

5.3 A Type *II* Sorting Circuit

This section sketches the proof of the following theorem, which improves upon the asymptotic result of Section 5.2, albeit at the expense of introducing a much larger multiplicative constant within the O-notation.

Theorem 4 Equation 1 holds.

The proof is the same as in Section 5.2, except that we strengthen the approximate sorting bound from Type *IV* to Type *II*. In other words, we claim that

$$|\texttt{Sort}(m, m^{\gamma}, m^{\gamma}, \mathcal{B})| = O(\log n) \quad (5)$$

for all $m \leq n$ and some positive constant γ, $0 < \gamma < 1$. We make use of three "gadgets" in order to establish the validity of Equation 5.

The first gadget is a circuit demonstrating that

$$|\texttt{Extract}(m, \frac{1}{4}\log m, m^{\alpha}, \mathcal{B})| = O(\log n)$$

for all $m \leq n$ and some positive constant α, $0 < \alpha < 1$. The main idea behind the design of this circuit is best understood by considering the special case where $m = n$ and m is a power of 2. In this case, the circuit will be nothing more than a butterfly tournament. By applying constant depth redundancy, we can assume that the fault probability ρ is a very small positive constant, say 0.01. Let S denote the set of $\frac{1}{4}\log m$ smallest inputs, and consider what happens to the elements of S as the m inputs pass throught the butterfly tournament. By the topology of the butterfly, no two members of S can meet each other more than once. Hence, each member of S will play at least $\frac{3}{4}\log m$ matches against players not belonging to S. By a standard Chernoff bound, one can show that with high probability every member of S will lose at least $\beta \log m$ of their $\frac{3}{4}\log m$ or more matches against non-S opponents, where β is some constant strictly greater than $1/2$. Note that the number of outputs losing more than $\beta\log m$ of their matches is less than m^{α} for some positive constant $\alpha < 1$ (see, for example, Exercise (9.42) of [6]).

For $m < n$, we make use of $r \stackrel{\text{def}}{=} \Theta(\log_m n)$ redundancy to drive the effective probability of failure of each "gate" (actually a sequence of r redundant gates) down to ρ^r. It is easily proven that this level of redundancy precisely compensates for the degradation in probability implied by the reduced number of matches (i.e., $\log m$ vs. $\log n$). The other detail to be taken care of concerns the requirement that m be a power of 2. We can handle this by arbitrarily partitioning the m inputs into groups whose sizes are powers of 2 corresponding to the binary representation of m. Each group of size greater than or equal to $m^{.9}$, say, can be handled as above. Finally, the "losers" from each of the larger groups, together with all of the elements from the smaller groups, can be combined to form the desired set of m^{α} outputs.

Our second gadget is a circuit that establishes

$$|\texttt{Extract}(m, m^{1-\delta}, m^{1-\delta/3}, \mathcal{B})| = O(\log n),$$

for all $m \leq n$ and where δ is an arbitray constant in $(0, 1)$. In the following discussion, we will assume that $m = n$ and that m is a power of 2. The remaining cases can be handled in much as the same way as in the construction of the first gadget (i.e., by applying $\Theta(\log_m n)$ redundancy, and by partitioning m according to its binary representation). The main building block of the circuit is a fault-tolerant ϵ-halver, where an *ϵ-halver* is a basic construct in the AKS sorting circuit.

An m-input ϵ-halver is a constant-depth comparator circuit with 2 sets of $m/2$ outputs labelled S (small) and L (large) such that at most ϵk of the k smallest (resp., largest) items are sent to L (resp., S), $1 \leq k \leq m/2$. Our fault-tolerant ϵ-halvers guarantee virtually the same property (with high probability); the only difference is that we require $k \geq \frac{1}{4}\log m$. As it happens, constructing such a fault-tolerant ϵ-halver is not a difficult task. In fact, our fault-tolerant ϵ-halvers are nothing more than ϵ'-halvers for some suitable constant $\epsilon' < \epsilon$. Note that explicit constructions of constant-depth expanders are known for any positive constant ϵ [2, 3, 11].

To build the second gadget, we apply an ϵ-halver to the entire set of m inputs, then apply two $m/2$-input ϵ-halvers to the resulting small and large halves, four $m/4$-input ϵ-halvers to the quarters, and so on. We continue this process until we have m^{δ} groups of size $m^{1-\delta}$ (i.e., we apply halvers to depth $\delta \log m$). For a sufficiently small choice of ϵ, at most $m^{\delta/2}$ of these groups will contain more than $\frac{1}{4}\log m$ of the smallest $m^{1-\delta}$ items (and we know which groups they are). The smallest items can then be isolated by removing the smallest $\frac{1}{4}\log m$ items from each of the other groups (using the first gadget) and moving them to join the items in the $m^{\delta/2}$ groups with potentially many small items.

Our third gadget establishes the bound of Equation 5. Once again, the essence of the construction is captured by considering the case $m = n$; for $m < n$, $\Theta(\log_m n)$ redundancy can be used to obtain the desired high probability bound. The idea is to run the AKS circuit as long as the upper bound on the size of the smallest set of strange elements in any bag is greater than $\frac{1}{4}\log m$. During these first $t \stackrel{\text{def}}{=} \Theta(\log m)$ steps, our fault-tolerant ϵ-halvers will function just as effectively as fault-free ϵ-halvers in terms of preserving the correctness of the algorithm. One may verify this claim by checking that the AKS proof of correctness does not require effective ϵ-halving of any set of size less than or equal to $\frac{1}{4}\log m$ during the first t steps.

After running the algorithm for t steps, the items that are badly out of position are contained in bags that are in the top 2/3 levels of the AKS tree, or are very strange items located in bags in the lowermost 1/3 of the AKS tree. The former items are few in number, and the latter items (also few in number) can be isolated using the second gadget. Note that the construction of the third gadget relies heavily on certain properties of the AKS sorting circuit. In verifying these properties, we have actually followed the construction of Paterson [12], rather than the original AKS paper.

6 Results for the Destructive Model

In this section, we analyze comparator circuits for which each comparator suffers a destructive fault with probability at most ρ. For the most part, we will assume that ρ is a small positive constant. We begin in Section 6.1 by proving an $\Omega(2^{-cn})$ lower bound for the failure probability of any `Maximum` circuit, where c is some positive constant. Then we prove that any circuit in the destructive model has to leave some item $\Omega(\log n)$ away from its correct position with (very) high probability. Most importantly, we will establish an $\Omega(\log^2 n)$ lower bound on the depth of any circuit that is capable of inserting an item into a sorted n-item list without moving any of the other items too far out of position. This lower bound will imply that any n-input approximate merger or approximate sorter must also have $\Omega(\log^2 n)$ depth.

The remainder of the section will focus on constructing destructive-fault-tolerant comparator circuits that perform certain tasks. We begin by analyzing the behavior of odd-even transposition sort in Section 6.2. Odd-even transposition sort will form a crucial component in all of our destructive-fault-tolerant merging and sorting circuits. For example, we use odd-even transposition sort to construct an $O(\log^2 n)$-depth Type I merging circuit and an $O(\log^3 n)$-depth Type I sorting circuit in Sections 6.3 and 6.4, respectively. Finally, we show how to construct a Type II sorting circuit with (optimal) depth $O(\log^2 n)$ in Section 6.5. The latter circuit combines the approximate merger described in Section 6.3 with recursive application of the AKS network.

6.1 Lower Bounds

The purpose of this section is to prove that destructive faults are much more difficult to tolerate than passive faults. Throughout this section, let $\mathcal{C} \stackrel{\text{def}}{=} (D, IV, \rho, \epsilon)$.

The following theorem proves a lower bound on the probability of failure of any circuit that solves an extremely simple special case of the maximum-finding problem.

Theorem 5 In order for $\texttt{Maximum}(n, n-1, \mathcal{C})$ to be solvable by any circuit with any depth, it must be that

$$\epsilon > 2^{-cn}$$

for some positive constant c depending on ρ, and n sufficiently large.

The next theorem establishes that, regardless of depth, any sorting circuit must err by $\Omega(\log n)$ on some output with extremely high probability. This motivates our decision to consider the problem of $O(\log n)$-approximate sorting. Our main lower bound in Theorem 7 will imply a bound on the depth of such a sorting circuit.

Theorem 6 There exist positive constants c and γ depending upon ρ such that for any circuit with any depth, the probability that one of the outputs is distance $c \log n$ away from the correct position is greater than $1 - 2^{-n^\gamma}$, n sufficiently large.

We will not present the proofs of the preceding pair of theorems. The basic idea is that we can make a claim which is even stronger than Theorem 5, but provable by induction. Then we can derive Theorem 6 by a careful application of Theorem 5.

Letting

$$\lambda = \left\lceil \log_4 \left(\frac{n-1}{2\Delta} \right) \right\rceil - 1 \tag{6}$$

and

$$\mu = \left\lfloor \log_\rho 2\epsilon \right\rfloor, \tag{7}$$

we have the following theorem.

Theorem 7 If $\rho \geq 2\epsilon$ then

$$|\texttt{Concatenate}(n, \Delta, \mathcal{C})| \geq \lambda\mu.$$

Before proving Theorem 7, we will discuss some of its consequences. The following special case is noteworthy.

Corollary 7.1 If $\epsilon = n^{-c}$ for some positive constant c, $\Delta = n^\gamma$ for some positive constant $\gamma < 1$, and ρ is a positive constant, then

$$|\texttt{Concatenate}(n, \Delta, \mathcal{C})| = \Omega(\log^2 n).$$

Furthermore, it is not difficult to prove that insertion, merging, and sorting circuits that handle most inputs are all at least as large as any circuit for the approximate concatenation problem. [The reason these facts are not completely trivial is that, for example, an approximate sorting circuit that only handles most inputs might well fail on the two input permutations of interest to the approximate concatenation problem.] In order to reduce concatenation to approximate sorting of most inputs, note that any n-input circuit that approximately sorts strictly more than half of all the $n!$ possible input permutations must approximately sort two permutations π and π' over $[0, n-1]$ such that $\pi'(i) = (\pi(i) + 1) \bmod n$, $0 \leq i < n$. It is straightforward to argue that such an approximate sorting circuit also solves the concatenation problem. The following corollary formally summarizes our results in this direction.

Corollary 7.2 If $\epsilon = n^{-c}$ for some positive constant c, $\Delta = n^\gamma$ for some positive constant $\gamma < 1$, and ρ is a positive constant, then

$$\begin{aligned} |\texttt{Insert}(n, \Delta, \mathcal{C})| &= \Omega(\log^2 n), \\ |\texttt{Merge}(n, n, \Delta, \mathcal{C})| &= \Omega(\log^2 n), \text{ and} \\ |\texttt{Sort}(n, \Delta, 0, \mathcal{C})| &= \Omega(\log^2 n). \end{aligned}$$

The remainder of this section is devoted to a proof of Theorem 7. We begin with several useful definitions.

An n-input, n-output comparator circuit of depth d (in terms of comparator gates) can be viewed as consisting of $d+1$ *levels* of n registers apiece (corresponding to the wires of the circuit). We will number these levels from 0 to d. Since there are two possible input permutations to the circuit (each one appearing with probability 1/2), and random comparator faults, the number received by any particular register is also random. For each register x, let X denote the random variable corresponding to the number received by that register, and let $label(x)$ in $[0, n-1]$ be chosen to satisfy

$$\begin{aligned} \Pr(X \geq label(x)) &\geq 1/2, \text{ and} \\ \Pr(X \leq label(x)) &\geq 1/2. \end{aligned}$$

Note that $label(x)$ need not be unique, and that the assignment of labels to registers depends upon the construction of the circuit.

Define register y to be a *descendant* of register x if and only if there is a non-zero chance that a number in register x could subsequently enter register y. Let us also define the *descendant set* $D(x,i)$ of register x to be the set of all descendants of register x at level i (we require that i be strictly greater than the level of x for $D(x,i)$ to be defined).

Note that the statement of the theorem implies that

$$\epsilon < 1/2, \tag{8}$$

and Equation 6 yields

$$(n-1)/4^{\lambda} > 2\Delta. \tag{9}$$

Definition 6.1 A register x at level $i\mu$, $i \geq 0$, is $(r_1, r_2)-$*bad* if and only if x will receive a number less than or equal to r_1 with probability greater than or equal to ϵ, and will receive an number greater than or equal to r_2 with probability greater than or equal to ϵ, where

$$r_2 - r_1 \geq (n-1)/4^i > 2\Delta.$$

Note that no output register of a concatenation circuit can be (r_1, r_2)-bad.

Definition 6.2 A descendant set $D(x, i\mu)$ is $(r_1, r_2)-$*bad* if and only if there are integers r_1 and r_2 satisfying

$$r_2 - r_1 \geq (n-1)/4^i > 2\Delta$$

and such that one of the following two conditions holds:

(i) register x will receive a number less than or equal to r_1 with probability greater than or equal to ϵ, and the label of every register in $D(x, i\mu)$ is greater than or equal to r_2, or

(ii) the label of every register in $D(x, i\mu)$ is less than or equal to r_1, and register x will receive a number greater than or equal to r_2 with probability greater than or equal to ϵ.

Since the two conditions of the preceding definition are entirely symmetric, we will not bother to deal with both cases in the arguments that follow.

We claim that a concatenation circuit of depth $i\mu$ cannot contain an (r_1, r_2)-bad descendant set $D(x, i\mu)$ for any register x. If it did, then with probability greater than or equal to ϵ, register x would receive a number greater than or equal to r_1. Note that any number that enters x must eventually enter some register in $D(x, i\mu)$. On the other hand, every register in $D(x, i\mu)$ must receive a number greater than or equal to r_2 with probability at least $1/2 > \epsilon$. Hence, every register y in $D(x, i\mu)$ must be be assigned an output rank (as required by the concatenation problem) greater than or equal to $r_2 - \Delta$. But if any register in $D(x, i\mu)$ receives a number less than or equal to r_1, then the circuit will fail since $(r_2 - \Delta) - r_1 > \Delta$. This proves the claim.

Lemma 6.1 Any comparator circuit of depth $i\mu$, $0 \leq i \leq \lambda$, contains either an (r_1, r_2)-bad leaf node or an (r_1, r_2)-bad descendant set $D(x, i\mu)$.

Proof: We prove the lemma by induction on i. For the base case, $i = 0$, note that some input register x receives 0 with probability 1/2, and $n-1$ with probability 1/2. Thus x is $(0, n-1)$-bad.

Now assume the lemma holds for $i = j$, $0 \leq j < \lambda$ and consider the induction step, namely $i = j+1$. Let $s = (n-1)/4^j$.

First note that if there exists a register x is at level $j\mu$, and a register y belonging to $D(x, (j+1)\mu)$, such that $|label(y) - label(x)| \geq s/4$ then y must be (r_1, r_2)-bad. To see this, assume without loss of generality that $label(y) - label(x) \geq s/4$. Note that y must receive a number less than or equal to $label(x)$ with probability at least $\rho^{\mu}/2$, which is at least ϵ by Equation 7. Furthermore, y must receive a number greater than or equal to $label(y)$ with probability at least ϵ. Hence, register y is $(label(x), label(y))$-bad.

By the argument of the preceding paragraph we now have the additional assumption that for every register x at level $j\mu$ and every register y belonging to $D(x, (j+1)\mu)$, $|label(y) - label(x)| < s/4$. Proceeding with the induction step, there are two cases to consider.

Case 1: There is an (r_1, r_2)-bad register x at level $j\mu$. Let S denote the set of labels associated with the registers in $D(x, (j+1)\mu)$. By our additional assumption, every pair of labels in S differ by at most $s/2$. Hence, either every label in S exceeds r_1 by at least $s/4$, or r_2 exceeds every label in S by at least $s/4$. In either case, $D(x, (j+1)\mu)$ represents a bad descendant set since $s/4 > 2\Delta$ by Equation 9.

Case 2: There is an (r_1, r_2)-bad descendant set $D(x, j\mu)$. We will assume that the first condition of the definition of an (r_1, r_2)-bad descendant set is satisfied; the other case can be handled by an entirely symmetric argument. Each register z in $D(x, (j+1)\mu)$ must be a descendant of some register y in $D(x, j\mu)$. By our additional assumption, *label*(y) can exceed *label*(z) by at most $s/4$. Furthermore, by the definition of an (r_1, r_2)-bad descendant set, *label*$(y) \geq r_2$. Hence, *label*$(z) \geq r_2 - s/4$, and $D(x, (j+1)\mu)$ must be $(r_1, r_2 - s/4)$-bad. □

The preceding lemma implies that for any concatenation circuit of depth $i\mu$ we must have $i \geq \lambda$. Finally, it is immediate that any concatenation circuit of depth d (not necessarily a multiple of μ) can be augmented to form a concatenation circuit of depth $\lceil d/\mu \rceil \mu$. Hence, the proof of Theorem 7 is complete.

6.2 Odd-Even Transposition Sort

In this section, we analyze the effect of random destructive faults on the performance of odd-even transposition sort. Our main result is stated below.

Theorem 8 For any constant δ, there exists a small constant ρ such that

$$|\texttt{Sort}(m, \delta m, 0, (D, I, \rho, 2^{-cm}))| = O(m)$$

for some positive constant c.

We will actually prove that the `Sort` problem of Theorem 8 can be solved by an odd-even transposition circuit with $O(m)$ depth. Before we sketch the proof, it is worth mentioning that this theorem is best possible in two senses. First, Theorem 5 implies that any such sorting circuit must have at least some exponentially small failure probability. Second, Theorem 6 demonstrates that if $\delta < 1/8$ then the circuit depth must be $\Omega(m)$ (the bound on δ is necessary to ensure that $\lambda \geq 1$).

We also point out that a similar Type I result holds for the passive model. In that model, we can get an exact sorting circuit instead of an approximate one, and the argument is significantly simpler. Using these results, we can easily show that other sorting algorithms which call odd-even transpostion as their main subroutine (shearsort, for example) can tolerate faults as well.

We will sketch the idea for showing that $9m$ items can be sorted to within $\frac{1}{2}m$ of the correct positions in $10m$ levels. A similar argument can be used to prove Theorem 8. Throughout, we will assume that ρ is an arbitrarily small constant. (By being more careful with the analysis, we can prove that a similar result is true with ρ at the level of 1%. We do not know if a similar result can be proved for ρ near 1/2 however.)

Without loss of generality, we will assume that the $9m$ inputs are 0's and 1's. In what follows, we will focus on the positions of the 0's within each level of the network as the 0's and 1's move forward through the network. Initially the 0's can be located anywhere among the $9m$ items, but at the end, we must show that the 0's are near the top. In particular, we will show that the ith 0 will be in position $i + \frac{1}{2}m$ or higher at the end (level $10m$) with high probability, no matter where it started at the input level. A similar analysis will be used to bound the final position of the 1's.

As the 0's move forward through the network they gradually move upward in the 1's. In particular, a 0 in position i which is compared to a 1 in position $i-1$ at level l will move to position $i-1$ after the comparison with probability $1-\rho$. Hence 0's are likely to move upward as they move forward. The problem in analyzing the movement of the 0's, however, is that they can block each other's upward movement. In particular, if one 0 moves the wrong way, it can cause a ripple effect much like a multicar collision on a highway. Since the probability of failures is constant, there are likely to be many such collisions. In addition to slowing things down, such collisions also introduce all sorts of dependence problems when one tries to analyze the probabilistic movement of each 0 through the network.

In order to get around these difficulties, we model the moves made by the 0's with a $9m \times 10m$ matrix $A = (a_{i,j})$ of random biased coins. In particular, $a_{i,j} = H$ with probability $1-\rho$. The coin in the ith row and jth column of the matrix will be used to determine whether or not the comparator entered by the ith 0 at time j is faulty (if there is one). (We number the 0's, starting at 1, from top to bottom at the outset, and we never alter the relative order of the 0's during the sorting.) Note that if two 0's enter the same comparator at some time j, the two associated coin flips could conflict in determining whether or not the comparator is faulty. However, we will be able to assume that comparisons between two 0's (or two 1's) are resolved according to the initial ordering of the 0's (1's); we do not need to refer to the coin flips in such cases.

In particular, if the ith 0 is compared to a 1 ahead in level t, then the 0 moves upward one position if and only if $a_{i,t} = H$. If the ith 0 is compared to a 1 behind in level t, then the 0 moves downward if and only if $a_{i,t} = T$. If the ith 0 is compared to a 0 ahead at level t (i.e., if it is blocked ahead by the $(i-1)$th 0, then the ith 0 stays in the same position, and we change the value of $a_{i,t}$ to be an X without checking to see if $a_{i,t} = H$. If the ith 0 is compared to a 0 behind at level t (i.e., if it is blocked from behind by the $(i+1)$st 0), then the ith 0 stays in the same position and we change the value of $a_{i,t}$ to be Y without checking to see if $a_{i,t} = T$.

After running the algorithm, the matrix A would now contain some X's and Y's. Call the new matrix $A^* = (a^*_{i,j})$. Let k denote the overall number of 0's. Define t_k to be the last time that the kth 0 was blocked by the $(k-1)$st 0. In other words, t_k is the maximum integer such that $a^*_{k,t_k} = x$. Next, define t_{k-1} to be the largest integer such that $t_{k-1} < t_k$ and $a^*_{k-1,t_{k-1}} = x$. In other words, t_{k-1} is the last time the $(k-1)$st 0 was blocked by the $(k-2)$nd 0 before the time that the $(k-1)$st 0 blocked the kth 0. Proceeding in a similar fashion, define t_j for $j = k-2, k-3, \cdots, 1$ to be the largest integer such that $t_j < t_{j+1}$ and $a^*_{j,t_j} = X$. (It may be that t_j is undefined for some j, in which case, we set $t_j = t_{j-1} = \cdots = t_1 = 0$.)

Let S denote the string of coins

$$a_{1,t_1+1}a_{1,t_1+2}\cdots a_{1,t_2-1}a_{2,t_2+1}a_{2,t_2+2}\cdots$$
$$a_{2,t_3-1}\cdots a_{k,t_k+1}a_{k,t_k+2}\cdots a_{k,10m}.$$

It is not difficult to show that S contains $10m-k-t$ coins. Let n_H denote the number of heads in S and n_T denote the number of tails in S.

Roughly speaking, the upward movement of the kth 0 is given by $n_T - n_H$. However, this bound is not precisely accurate because of boundary effects (i.e., caused by the 0's piling up against the boundary of the circuit). In fact, we can show that if $n_T < \frac{1}{4}m$, then the kth 0 will be within $\frac{1}{2}m$ of the correction position after sorting.

If the kth (i.e., the last) 0 is within $\frac{1}{2}m$ of its correct position, then so is every other 0. Hence, for some 0 to be more than $\frac{1}{2}m$ away from its correct final position, we need $n_T > \frac{1}{4}m$. Since the string of coins that comprises S forms a monotonically nonincreasing sequence of positions in A, A must have a monotonically nonincreasing sequence of $\frac{1}{4}m$ tails in order for n_T to be at least $\frac{1}{4}m$.

The analysis is conducted by showing that with probability $1-2^{-cm}$ (for some positive constant c), A does not have a monotonically nonincreasing subsequence of $\frac{1}{4}m$ tails, provided that ρ is a sufficiently small constant. The details of the argument will be included in the final version of the paper.

6.3 A Type *I* Merging Circuit

We will now make use of the odd-even transposition result of the preceding section to obtain a Type *I* merging circuit. In particular, we prove the following corollary to Theorem 8.

Corollary 8.1

$$\texttt{Merge}(m_1, m_2, \Delta, (D, I, \rho, n^{-\alpha}))| = O(\log m \log n)$$

for some constant α, where $m \stackrel{\text{def}}{=} \max\{m_1, m_2\} = \Omega(\log n)$ and $\Delta = O(\log n)$.

Notice that insertion is a special case of merging, so that we can solve insertion in the same depth. Furthermore, the results of Section 6.1 imply that these bounds are tight.

When $m_1 = m_2$, the circuit is constructed recursively as follows. Given two m-items lists L_1 and L_2, define $L_{1\alpha}$ and $L_{2\alpha}$ to consist of the even-index elements of L_1 and L_2 (resp.), and define $L_{1\beta}$ and $L_{2\beta}$ to consist of odd-index elements of L_1 and L_2. If L_1 and L_2 are approximately sorted, then so are $L_{1\alpha}$, $L_{2\alpha}$, $L_{1\beta}$ and $L_{2\beta}$. Next recursively merge $L_{1\alpha}$ with $L_{2\beta}$ to form L_α, and $L_{1\beta}$ with $L_{2\alpha}$ to form L_β. By definition, L_α and L_β are approximately sorted. Next, shuffle L_α with L_β to form a list L' that is $2\log n$-approximately sorted. Finally, partition L' into contiguous blocks consting of $9\log n$ items each, and apply $10\log n$ levels of odd-even transposition sort to each block. Finish up by repartitioning the resulting list into contiguous blocks of size $9\log n$ so that the boundaries of the earlier blocks fall in the center of the current blocks, and apply $10\log n$ levels of odd-even transposition sort once again. With high probability, the resulting list will be sorted to within Δ. The details of the proof will be included in the final version of the paper.

When $m_1 > m_2$, we can approximately merge the two lists by artificially extending the shorter list with $-\infty$ items. The details of the construction will be included in the final version of the paper.

6.4 A Type *I* Sorting Circuit

As an immediate consequence of the merging circuit just described, we can build a circuit with depth $O(\log^2 m \log n)$ that will approximately sort all m-item input permutations with high probability given random destructive failures. As in Batcher's odd-even merge algorithm [4], we simply partition the m items to be sorted into two subset of $m/2$ items each, sort the subsets recursively, and then merge them.

6.5 A Type *II* Sorting Circuit

Throughout this section, let

$$\mathcal{D} = (D, II, \rho, n^{-c})$$

where c is an arbitrary positive constant and ρ is a sufficiently small positive constant.

We do not know whether or not there is a destructive-fault-tolerant circuit with depth $o(\log^3 n)$ that is capable of approximately sorting all n-input permutations. We can prove the following Type *II* result, however.

Theorem 9 Equation 2 holds.

The structure of the proof is quite similar to that of Section 5.3. In this case, we will set ξ to $1/\sqrt{\log n}$ in Equation 3. We can afford to apply the $O(\log^2 m \log n)$-depth, Type *I* sorting circuit of Section 6.4 at this cut-off point.

To complete the argument, we can make use of the merging result of Section 6.3 to perform insertion, and all that remains is to prove that

$$|\texttt{Sort}(m, m^\gamma, m^\gamma, \mathcal{D})| = O(\log m \log n) \qquad (10)$$

for all $m \le n$, any positive constant c, and some positive constant γ, $0 < \gamma < 1$. This bound will be established by defining three gadgets very similar to those of Section 5.3. We sketch the main differences below. More details will be included in the final version of the paper.

The first gadget gives

$$|\texttt{Extract}(m, \log m, 2\log m, \mathcal{D})| = O(\log m \log n)$$

for all $m \leq n$ and some positive constant α, $0 < \alpha < 1$. The circuit is constructed recursively, using odd-even transposition sort at each step of the recursion.

The second gadget establishes the bound

$$|\texttt{Extract}(m, m^{1-\delta}, m^{1-\delta/3}, \mathcal{D})| = O(\log m \log n),$$

for all $m \leq n$ and where δ is an arbitray constant in $(0, 1)$. The circuit is very similar to that defined in Section 5.3. The main difference is that we cannot take advantage of the redundancy trick; as a result, our destructive-fault-tolerant ϵ-halvers require $k \geq \log n$ (as opposed to $k = \Omega(\log m)$). This would be problematic were it not for the fact that we cut off the recursion before any subproblems involving small values of m begin to arise.

The structure of the third gadget is virtually unchanged from Section 5.3, and establishes the bound of Equation 10. In this case, we run AKS as long the upper bound on the size of the smallest set of strange elements in any bag is greater than $\log n$.

7 Final Remarks

We have used some of the ideas underlying the Type *IV* bound of Section 5.2 to design sorting circuits that are smaller than the classic Batcher circuits, and that are also passive-fault-tolerant. For example, we have constructed a 1024-input circuit that correctly sorts 99% of input permutations within 48 levels if there are no faults. If $\rho = .01$ (i.e., if 1% of the comparators sustain passive faults), then the same circuit correctly sorts 91% of all input permutations within 49 levels, and if $\rho = .02$, the circuit correctly sorts 91% of input permutations within 50 levels. (With $\rho = .05$, we can sort 80% of input permutations within 60 levels.) These results compare favorably to the best known deterministic sorting circuits. For example, the best previously known circuit for sorting 1024 inputs has depth 55, and has essentially no fault-tolerance.

References

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

[2] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6:83–96, 1986.

[3] S. Assaf and E. Upfal. Fault tolerant sorting network. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 275–284, October 1990.

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

[5] U. Feige, D. Peleg, P. Raghavan, and E. Upfal. Computing with unreliable information. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 128–137, May 1990.

[6] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading, MA, 1989.

[7] D. J. Kleitman, A. R. Meyer, R. L. Rivest, J. Spencer, and K. Winklemann. Coping with errors in binary search procedures. *JCSS*, 20:396–404, 1980.

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

[9] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes*. Morgan-Kaufmann, San Mateo, CA, 1991.

[10] F. 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 264–274, October 1990.

[11] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanugan conjecture and explicit construction of expanders. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, May 1986.

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

[13] A. Pelc. Searching with known error probability. *Theoretical Computer Science*, 63:185–202, 1989.

[14] L. Rudolph. A robust sorting network. *IEEE Transactions on Computers*, C–34:326–335, 1985.

[15] A. C. Yao and F. F. Yao. On fault-tolerant networks for sorting. *SIAM J. Comput.*, 14:120–128, 1985.