HYPERCUBIC SORTING NETWORKS*

TOM LEIGHTON† AND *C. GREG PLAXTON*‡

Abstract. This paper provides an analysis of a natural d-round tournament over $n = 2^d$ players, and demonstrates that the tournament possesses a surprisingly strong ranking property. The ranking property of this tournament is used to design efficient sorting algorithms for a variety of different models of parallel computation:

(i) a comparator network of depth $c \cdot \lg n$, $c \approx 7.44$, that sorts the vast majority of the $n!$ possible input permutations,
(ii) an $O(\lg n)$-depth hypercubic comparator network that sorts the vast majority of permutations,
(iii) a hypercubic sorting network with nearly logarithmic depth,
(iv) an $O(\lg n)$-time randomized sorting algorithm for any hypercubic machine (other such algorithms have been previously discovered, but this algorithm has a significantly smaller failure probability than any previously known algorithm), and
(v) a randomized algorithm for sorting n $O(m)$-bit records on an $(n \lg n)$-node omega machine in $O(m + \lg n)$ bit steps.

Key words. parallel sorting, sorting networks, hypercubic machines

AMS subject classifications. 68P10, 68Q22, 68Q25, 68R05

1. Introduction. A *comparator network* is an n-input, n-output acyclic circuit made up of wires and 2-input, 2-output comparator gates. The input wires of the network are numbered from 0 to $n-1$, as are the output wires. The input to the network is an integer vector of length n, where the ith component of the vector is received on input wire i, $0 \le i < n$. The two outputs of each comparator gate are labeled "min" and "max", respectively, while the two inputs are not labeled. On input x and y, a comparator gate routes $\min\{x, y\}$ to its "min" output and routes $\max\{x, y\}$ to its "max" output. It is straightforward to prove (by induction on the depth of the network) that any comparator network induces some permutation of the input vector on the n output wires. We say that a given comparator network *sorts* a particular vector if and only if the value routed to output i is less than or equal to the value routed to output $i+1$, $0 \le i < n-1$.

An n-input comparator network is a *sorting network* if and only if it sorts every possible input vector. It is straightforward to prove that any n-input comparator network that sorts the $n!$ permutations of $\{0, \ldots, n-1\}$ is a sorting network. In fact, any n-input comparator network that sorts the 2^n possible 0-1 vectors of length n is a sorting network. The latter result is known as the 0-1 principle for sorting networks [11, §5.3.4].

---

* This paper combines results appearing in preliminary form as A (fairly) Simple Circuit that Usually Sorts, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 264-274; and as A Hypercubic Sorting Network with Nearly Logarithmic Depth, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 405-416.

† Department of Mathematics and Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139. Email: `ftl@math.mit.edu`. Supported by AFOSR Contract F49620–92–J–0125, DARPA Contract N00014–91–J–1698, and DARPA Contract N00014–92–J–1799.

‡ Department of Computer Science, University of Texas at Austin, Austin, TX 78712. Email: `plaxton@cs.utexas.edu`. Supported by NSF Research Initiation Award CCR–9111591, and the Texas Advanced Research Program under Grant No. 003658–480. Part of this work was done while the author was visiting the MIT Laboratory for Computer Science.

It is natural to consider the problem of constructing sorting networks of optimal depth. Note that at most $\lfloor n/2 \rfloor$ comparisons can be performed at any given level of a comparator network. Hence the well-known $\Omega(n \lg n)$ sequential lower bound for comparison-based sorting implies an $\Omega(\lg n)$ lower bound on the depth of any n-input sorting network. An elegant $O(\lg^2 n)$-depth upper bound is given by Batcher's bitonic sorting network [4]. For small values of n, the depth of bitonic sort either matches or is very close to matching that of the best constructions known (a very limited number of which are known to be optimal) [11, §5.3.4]. Thus, one might suspect the depth of Batcher's bitonic sorting network to be optimal to within a constant factor, or perhaps even to within a lower-order additive term. Consider Knuth's Exercise 5.3.4.51 [11] (posed as an open problem): Prove that the asymptotic value of $\hat{S}(n)$ is not $O(n \cdot \lg n)$, where $\hat{S}(n)$ denotes the minimal size (number of comparator gates) of an n-input sorting network of *any* depth. The source of the difficulty of this particular exercise was subsequently revealed by Ajtai, Komlós, and Szemerédi [2], who provided an optimal $O(\lg n)$-depth construction known as the *AKS sorting network*.

While the AKS sorting network represents a major theoretical breakthrough, it suffers from two significant shortcomings. First, the multiplicative constant hidden within the O-notation is sufficiently large that the result remains impractical. Second, the structure of the network is sufficiently "irregular" that it does not seem to map efficiently to common interconnection schemes. In fact, Cypher has proven that any emulation of the AKS network on the cube-connected cycles requires $\Omega(\lg^2 n)$ time [7]. The latter issue is of significant interest, since a primary motivation for considering the problem of constructing small-depth sorting networks is to obtain a fast parallel sorting algorithm for a general-purpose parallel computer. In other words, it would be highly desirable to identify a small-depth sorting network that could be implemented efficiently on a topology that is also useful for performing operations other than sorting.

In this paper we pursue a new approach to the problem of designing small-depth sorting networks with "regular" structure. Our notion of regularity is enforced by restricting the set of permutations that can be used to connect successive levels of gates in a comparator network. In particular, we say that a comparator network is *hypercubic* if and only if successive levels are connected either by a shuffle or an unshuffle (inverse shuffle) permutation. (These terms are defined more precisely in §3.) Knuth's Exercise 5.3.4.47 [11], posed as an open problem, may be viewed as asking for the depth complexity of *shuffle-only sorting networks*, in which every pair of adjacent levels is connected by a shuffle permutation. Batcher's bitonic sort provides an $O(\lg^2 n)$ upper bound for this problem, and recently, Plaxton and Suel [17] have established an $\Omega(\lg^2 n / \lg \lg n)$ lower bound. (The same lower bound holds for the class of unshuffle-only sorting networks.)

From a practical point of view, Knuth's shuffle-only requirement would seem to be overly-restrictive. It is motivated by a certain correspondence between hypercubic comparator networks and the class of *hypercubic machines* (e.g., the hypercube, butterfly, cube-connected cycles, omega, and shuffle-exchange). This correspondence allows any shuffle-only comparator network to be efficiently emulated (i.e., with constant slowdown) on any hypercubic machine. (We remark that "hypercubic machines" are more commonly referred to as "hypercubic networks" [12, Chapter 3]. We prefer the term "hypercubic machines" in the present context only because we use the term "networks" to refer to comparator networks.) However, the class of hypercubic machines is most often characterized in terms of efficient emulation of so-called "normal" hypercube algorithms [12, Chapter 3], which effectively allow the data to either be shuffled or unshuffled at each step. (More formally, a hypercube algorithm is "normal" if it satisfies the following two conditions: (i) in any given step of the computation, communication occurs across a single dimension, and (ii) in any pair of successive steps, communication occurs across an adjacent pair of dimensions.) Thus, hypercubic comparator networks, as defined above, would seem to represent the most natural class of comparator networks corresponding to hypercubic machines.

Our approach to the design of efficient hypercubic sorting networks is based on the following *d-round no-elimination tournament* defined over $n = 2^d$ players, $d \geq 0$. For $d = 0$, the tournament has 0 rounds; no matches are played. For $d > 0$, $n/2$ matches are played in the first round according to an arbitrary pairing of the n players. The next $d-1$ rounds are defined by recursively running a no-elimination tournament amongst the $n/2$ winners, and (in parallel) a disjoint no-elimination tournament amongst the $n/2$ losers. (We have chosen to call this a "no-elimination" tournament in order to contrast it with the more usual "single-elimination" or "double-elimination" formats in which a player drops out of the tournament after suffering one or two losses.)

After a no-elimination tournament has been completed, each player has achieved a unique sequence of match outcomes (wins and losses, 1's and 0's) of length d. Let player i be the player that achieves a win-loss sequence corresponding to the d-bit number i; for example, in a 5-round tournament the sequence `WLLWL` would correspond to $i = 10010_2 = 18$. Assume that the outcomes of all matches are determined by an underlying total order. Further assume that there are n distinct amounts of prize money available 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 $n-1 = \mathtt{W}^d$, who is guaranteed to be the best player. Similarly, the smallest prize should be awarded to player $0 = \mathtt{L}^d$. On the other hand, it is not clear how to rank the remaining $n-2$ win-loss sequences. For instance, in an 8-round tournament, 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 §5, however, this is not true for the sequences `WLWLLWLL` and `LLLWWWWW`. 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.

The analysis of §5 not only implies that `WLWLLWLL` is a better record than `LLLWWWWW`, but also provides an efficient algorithm for computing a fixed permutation π of $\{0, \ldots, n-1\}$ such that with probability at least $1 - 2^{-n^{\varepsilon}}$, for some constant $\varepsilon > 0$, the actual rank of all but a small, fixed subset of the players is well-approximated by $\pi(i)$, $0 \leq i < n$. (See Theorem 5.1 for a more precise formulation of this result.)

Why does the no-elimination tournament 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 no-elimination tournament has the property that when two players meet in the ith round, they have achieved the same sequence of outcomes in two independent no-elimination 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.

The remainder of the paper is organized as follows. In §2, we discuss our applications of the no-elimination tournament. In §3, we provide a number of definitions. In §4, we present several basic lemmas. In §5, we analyze the sorting properties of the no-elimination tournament. Note that §5.3 contains a number of important technical definitions related to the no-elimination tournament. In §6 through §11, we present the applications of the no-elimination tournament discussed in §2. In §12, we offer some concluding remarks.

2. Overview of Applications. In §6 through §11 of the paper, we use the strong ranking property of the no-elimination tournament to design efficient sorting algorithms for a variety of different models of parallel computation. Most of our results are probabilistic in nature; for such results, the success probability is expressed in the form

$$1 - 2^{-2^{f(d)}},$$

for some function $f(d)$. (The parameter d is equal to $\lg n$, where n is the input size.) For the purposes of this introduction, it will be convenient to define a number of substantially different levels of "high probability" in terms of the function $f(d)$. Let us say that an event occurs *with very high probability* if $f(d) = \lg d + O(1)$, *with very*^{2} *high probability* if $f(d) = \Theta(\sqrt{d})$, with *very*^{3} *high probability* if

$$f(d) = \frac{d}{2^{\Theta(\sqrt{\lg d})}}$$

with very^4 *high probability* if $f(d) = \Theta\left(\frac{d}{(\lg d)\cdot\lg^* d}\right)$, and with *very*^{5} *high probability* if $f(d)$ can be set to any function that is $o(d)$. Note that an event occurs with very high probability if and only if the corresponding failure probability is polynomially small in terms of n. As it happens, all of the main probabilistic claims made in this paper hold with very^2 high probability or better. We have defined the very high probability threshold only for the purpose of contrasting the results of §10 with those of previous authors.

We now survey the applications of §6 through §11. In §6, we define a comparator network of depth $c\cdot\lg n$, $c \approx 7.44$, that sorts a randomly chosen input permutation with very^5 high probability (see Theorem 6.1). (We remark that this comparator network is not hypercubic. A hypercubic version of the construction is discussed in the next paragraph.) At the expense of allowing the network to fail on a small fraction of the $n!$ possible input permutations, this construction improves upon the asymptotic depth of the best previously known sorting networks by several orders of magnitude [2, 15]. We make use of the AKS construction as part of our network. However, the use of the AKS construction can be avoided at the expense of decreasing the success probability from very^5 to very^3 high. (The depth bound remains unchanged.) The topology of our very^3 high probability network is quite simple and does not make use of expanders.

In §7, we present a hypercubic version of the construction of §6. In particular, we define an $O(\lg n)$-depth hypercubic comparator network that sorts a randomly chosen input permutation with very^3 high probability (see Theorem 7.1). We have not calculated the constant factor within the $O(\lg n)$-depth bound, which is moderately larger than the constant of approximately 7.44 associated with our non-hypercubic construction.

In §8 and §9, we provide a general method for constructing a sorting network from a comparator network that sorts most permutations. More specifically, §8 describes how to construct a (hypercubic) high-order merging network from a (hypercubic) comparator network that sorts most input permutations. In §9, we make use of a hypercubic high-order merging network to develop a recurrence for the depth complexity of hypercubic sorting networks. The analysis of this recurrence, presented in Appendix A, yields the main non-probabilistic claim of our paper, namely, that there exist hypercubic sorting networks of depth

$$2^{O\left(\sqrt{\lg\lg n}\right)} \cdot \lg n.$$

Note that this bound is $o(\lg^{1+\varepsilon} n)$ for any constant $\varepsilon > 0$. (See Theorem 9.1 for a more precise form of the upper bound.) Given the aforementioned $\Omega(\lg^2 n/\lg\lg n)$ lower bound of Plaxton and Suel [17], our upper bound establishes a surprisingly strong separation between the power of shuffle-only comparator networks and that of hypercubic comparator networks.

Unfortunately, each of the network constructions given in §6, §7, and §9 is non-uniform in the following sense: No deterministic polynomial-time algorithm is known for generating the family of networks for which existence has been established. On the positive side, existence of randomized polynomial-time generation algorithm for each of these network families is a straightforward consequence of our results.

In §10, an optimal $O(\lg n)$-time randomized sorting algorithm is given for any hypercubic machine. The algorithm runs in $O(\lg n)$ time on every input permutation with very^4 high probability, and uses only $O(1)$ storage at each processor. Furthermore, a very^2 high probability version of the algorithm never has more than 2 records at the same processor (where the "2" is only necessary for implementing compare-interchange operations), and requires essentially no auxiliary variables. (A global OR operation involving a single bit at each processor is used to check whether the sort has been completed.) A number of optimal-time randomized sorting algorithms were previously known for certain hypercubic machines. For example, the Flashsort algorithm of Reif and Valiant [19] is in this category. However, none of these algorithms has a success probability better than "very high". Probability of failure aside, Flashsort requires more storage than our algorithm, since it makes use of a $\Theta(\lg n)$-sized priority queue at each processor. On the other hand, a very high probability sorting algorithm with constant size queues has previously been given by Leighton, Maggs, Ranade, and Rao [13]. Like Batcher's $O(\lg^2 n)$ bitonic sorting algorithm, the very^2 high probability version of 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. (The very^4 high probability version is adaptive because it makes use of the Sharesort algorithm of Cypher and Plaxton as a subroutine [9].)

Note that the permutation routing problem, in which each processor has a packet of information to send to another processor, and no two packets are destined to the same processor, is trivially reducible to the sorting problem. (The idea is to sort the packets based on their destination addresses.) Hence, our sorting bounds also apply to that fundamental routing problem. In fact, standard reductions [12, §3.4.3] allow us to apply our sorting algorithm to efficiently solve a variety of other routing problems as well (e.g., many-to-one routing with combining). Interestingly, all previously known optimal-time algorithms for permutation routing on hypercubic machines [13, 18, 20] are randomized, and do not achieve a success probability better than "very high". Thus, the results of §10 provide a permutation routing algorithm for

TABLE 1
Type conventions.

Symbol	*Type*	*Symbol*	*Type*
a, b, d, i, j, k, m, n	integer	$\mathcal{N}$	comparator network
c	real constant	α, β	binary string
f, g, h	function	ϵ	empty string
p, q	real number in $[0, 1]$	π	permutation
u, v, w, z	real number	Π	set of permutations
x, y	various	ϕ	0-1 vector
A, B, C	set	Φ	set of 0-1 vectors
E	probabilistic event	$o, \omega, \Theta, \Omega$	asymptotic symbol
X, Y	random variable	Σ	summation symbol
$\mathcal{D}$	probability distribution	$\gamma_c, \mu_c, \upsilon_c$	defined constant
$\mathcal{M}$	parallel machine	other Greek letters	real number/function

hypercubic machines with a much smaller probability of failure than any previously known $O(\lg n)$-time algorithm.

Our final application is described in §11, where we give a randomized algorithm for sorting n $O(m)$-bit records on an $(n \cdot \lg n)$-node omega machine in $O(m + \lg n)$ bit steps with very^2 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 (assuming, as is typical, that $m = \Omega(\lg n)$). The only previous result of this kind that does not rely on the AKS sorting network is the recent work of Aiello, Leighton, Maggs, and Newman [1], which provides a randomized bit-serial *routing* algorithm that runs in optimal time with very high probability on the hypercube. That paper does not address either the combining or sorting problems, however, and does not apply to any of the bounded-degree hypercubic machines (e.g., the butterfly, cube-connected cycles, omega, and shuffle-exchange). All previously known algorithms for routing and sorting on bounded-degree hypercubic machines, and for sorting on the hypercube, require $\Omega(\lg^2 n)$ bit steps.

A defect of the randomized sorting algorithms described in §10 and §11 is that each requires a table of permutation information to be precomputed and stored in the nodes of the machine before the algorithm is executed. Fortunately, this defect may be viewed as a relatively minor one since: (i) the table only needs to be computed once for a given machine size n (i.e., the same table can be used to sort all $n!$ possible input permutations in the time bounds stated above), (ii) the table only occupies a constant number of words per machine node, and (iii) there is a deterministic polynomial-time (in n) algorithm for computing the table. For the purposes of the present paper, it is convenient to define such a "table-based" randomized sorting algorithm as *polynomial-time uniform* if and only if it satisfies properties (i), (ii), and (iii) above. All of the randomized sorting algorithms presented in this paper are polynomial-time uniform. (In fact, the tables used by our algorithms can easily be computed in **NC**.)

3. Definitions. In the sections that follow, we present basic definitions related to notational conventions, vectors, permutations, 0-1 vectors, (hypercubic) comparator networks, randomness, network composition, and network families. A number of definitions related to our analysis of the 0-1 no-elimination tournament are postponed to §5.

3.1. Notational Conventions. Our type conventions and defined constants are summarized in Tables 1 and 2, respectively. (We remark that primed and subscripted variables have the same type as their unprimed and unsubscripted counterparts.)

The functions $\lg x$ and $\mathrm{pow}(x)$ denote $\log_2 x$ and 2^x, respectively.

For all nonnegative integers a and i, let a_i denote bit i in the binary representation of a. (Bit 0 is the least significant bit.)

3.2. Vectors. A *d-vector*, $d \geq 0$, is an integer vector of length $\mathrm{pow}(d)$. For any d-vector x, we index the components of x from 0 through $\mathrm{pow}(d) - 1$, and denote the ith component $x(i)$.

A d-vector x is *sorted* if and only if $x(i) \leq x(i+1)$, $0 \leq i < \mathrm{pow}(d) - 1$.

For any d-vector x, the ith *a-cube* of d-vector x, $0 \leq a \leq d$, $0 \leq i < \mathrm{pow}(d-a)$, is the a-vector y such that $y(j) = x(i \cdot \mathrm{pow}(a) + j)$, $0 \leq j < \mathrm{pow}(a)$.

3.3. Permutations. A *permutation* π of length k, $k \geq 0$, is a vector of length k satisfying the following condition: For each i, $0 \leq i < k$, there is a j, $0 \leq j < k$, such that $\pi(j) = i$. If length-k permutation π is applied to length-k vector x, the resulting length-k vector x' is such that $x'(\pi(i)) = x(i)$, $0 \leq i < k$.

For all length-k permutations π and π', the length-k permutation obtained by applying π to π' is denoted $\pi \circ \pi'$.

A *d-permutation*, $d \geq 0$, is a permutation of length $\mathrm{pow}(d)$.

Let $\Pi(d)$ denote the set of all $\mathrm{pow}(d)!$ d-permutations.

For $0 \leq a \leq d$, let $\Pi(d, a)$ denote the $\mathrm{pow}(a)!$ d-permutations π in $\Pi(d)$ such that: (i) π permutes within a-cubes, and (ii) π applies the same a-permutation within each a-cube.

The *shuffle d-permutation*, denoted $\hookleftarrow_d$, has ith component $i_{d-2} \cdots i_0 i_{d-1}$, $0 \leq i < \mathrm{pow}(d)$. The *$k$-shuffle d-permutation*, denoted $\hookleftarrow_d^k$ is the d-permutation obtained by composing k shuffle d-permutations.

The *unshuffle d-permutation*, denoted $\hookrightarrow_d$, has ith component $i_0 i_{d-1} \cdots i_1$, $0 \leq i < \mathrm{pow}(d)$, and is equal to the inverse of the shuffle d-permutation. Thus $\hookrightarrow_d = \hookleftarrow_d^{-1}$. The *$k$-unshuffle d-permutation*, denoted $\hookrightarrow_d^k$, is the d-permutation obtained by composing k unshuffle d-permutations. Note that $\hookleftarrow_d^k = \hookrightarrow_d^{-k}$ for all k.

3.4. 0-1 Vectors. A *0-1 d-vector* is a d-vector over $\{0, 1\}$. Let $\Phi(d)$ denote the set of all $\mathrm{pow}(\mathrm{pow}(d))$ 0-1 d-vectors.

For $0 \leq k \leq \mathrm{pow}(d)$, let $\Phi(d, k)$ denote the set of all

$$\binom{\mathrm{pow}(d)}{k}$$

0-1 d-vectors with k 0's and $(\mathrm{pow}(d) - k)$ 1's.

A 0-1 d-vector is *trivial* if and only if it belongs to $\Phi(d, 0) \cup \Phi(d, \mathrm{pow}(d))$. (Otherwise, it is *non-trivial*.)

For any d-permutation π, and all k such that $0 \leq k \leq \mathrm{pow}(d)$, we define the *$k$th 0-1 d-vector corresponding to d-permutation π*, denoted ϕ_π^k, as follows:

$$\phi_\pi^k(i) = \begin{cases} 0 & \text{if } 0 \leq \pi(i) < k, \\ 1 & \text{if } k \leq \pi(i) < \mathrm{pow}(d). \end{cases}$$

Note that ϕ_π^k belongs to $\Phi(d, k)$.

For any d-permutation π, let $\Phi_\pi = \cup_{0 \leq k \leq \mathrm{pow}(d)} \phi_\pi^k$.

TABLE 2
Constants.

Symbol	*Constant*
e	2.7182818...
γ_c	see Equation (7)
μ_c	see Equation (8)
υ_c	see Equation (9)

Let ϕ be a 0-1 d-vector, i be the maximum index for which $\phi(i) = 0$ (or -1 if ϕ belongs to $\Phi(d, 0)$), and j be the minimum index for which $\phi(j) = 1$ (or $\text{pow}(d)$ if ϕ belongs to $\Phi(d, \text{pow}(d))$). We say that ϕ has a *dirty region of size* $i-j+1$ corresponding to the sequence of components $\langle \phi(j), \ldots, \phi(i) \rangle$. Observe that ϕ is sorted if and only if $i = j - 1$. (Thus, the dirty region of a sorted 0-1 vector is defined to be empty, and has size 0.)

A 0-1 d-vector is a-*sorted*, $0 \leq a \leq d$, if and only if it has a dirty region of size at most $\text{pow}(a)$.

For nonnegative integers a and b, let $\Phi_M(a, b)$ denote the set of all 0-1 $(a + b)$-vectors ϕ such that every a-cube of ϕ is sorted.

We remark that if 0-1 d-vector ϕ is a-sorted, then the 0-1 d-vector ϕ' obtained by applying $\hookrightarrow_d^a$ to ϕ belongs to $\Phi_M(d - a, a)$. Furthermore, each $(d - a)$-cube of ϕ' has the same number of 0's to within 1.

3.5. (Hypercubic) Comparator Networks. This paper studies the depth complexity of certain classes of comparator networks. For the sake of brevity, we will use the term "network" to mean "comparator network" throughout the remainder of the paper.

For nonnegative integers a and d, a *depth-a d-network* consists of a disjoint *levels*, numbered from 0 to $a-1$, each of which has $\text{pow}(d)$ associated *input* and *output wires*. (Note that every depth-0 d-network is the empty network.) The input and output wires of each level are numbered from 0 to $\text{pow}(d) - 1$. Output wire j on level i and input wire j on level $i + 1$ represent the same wire, $0 \leq i < a - 1$, $0 \leq j < \text{pow}(d)$. The level 0 input wires (resp., level $a - 1$ output wires) of a given network $\mathcal{N}$ are also referred to as the input wires (resp., output wires) of $\mathcal{N}$.

In order to complete our definition of a network, it remains only to define the structure and behavior of a single level. If $d = 0$, each level consists of a single wire, and the lone input is passed directly to the output. For $d > 0$, each level consists of two phases: a permutation phase followed by an operation phase.

In the permutation phase, some d-permutation π is applied to the $\text{pow}(d)$ input wires of the level. We refer to the resulting ordered set of $\text{pow}(d)$ wires as the *intermediate wires* of the level. In an execution of the permutation phase, the values received by the input wires are passed to the intermediate wires according to d-permutation π: Intermediate wire $\pi(j)$ receives its value from input wire j, $0 \leq j < \text{pow}(d)$.

In the operation phase, the values carried by the $\text{pow}(d)$ intermediate wires are passed through a set of $\text{pow}(d - 1)$ 2-input, 2-output *gates*, numbered from 0 to $\text{pow}(d - 1) - 1$. Intermediate wires (resp., output wires) $2 \cdot j$ and $2 \cdot j + 1$ are input to (resp., output from) the jth gate of the level. There are five kinds of gates in our d-networks: "0", "1", "+", "$-$", and "?". The action of each of these gates is described below.

"0": On input (x, y), a "0" gate produces output (x, y).

"1": On input (x, y), a "1" gate produces output (y, x).
"+": On input (x, y), a "+" gate produces output $(\min\{x, y\}, \max\{x, y\})$.
"−": On input (x, y), a "−" gate produces output $(\max\{x, y\}, \min\{x, y\})$.
"?": On input (x, y), a "?" gate produces output (x, y) with probability $1/2$, and output (y, x) with probability $1/2$. This gate is only used in §10 and §11 of the paper.

A d-network is *hypercubic* if and only if the d-permutation applied in each level of the d-network is either $\hookleftarrow_d$ or $\hookrightarrow_d$.

3.6. Randomness. A d-network $\mathcal{N}$ is *deterministic* if and only if $\mathcal{N}$ satisfies the following conditions: (i) the d-permutation applied in the permutation phase of each level is fixed, (ii) the type of each gate is fixed, and (iii) no gate is of type "?".

In general, we allow our d-networks to be random. A depth-a d-network $\mathcal{N}$ is *random* if and only if $\mathcal{N}$ is given by some fixed probability distribution over the set of all deterministic depth-a d-networks. (Each time an input vector is passed to a random network $\mathcal{N}$, the network behaves as a randomly chosen deterministic network drawn from the distribution defining $\mathcal{N}$.)

We have introduced the notion of a random network primarily as a technical convenience, since the random aspects of any construction can be eliminated using Lemma 4.8. Unfortunately, reliance on Lemma 4.8 leads to network constructions that are not polynomial-time uniform.

In §10 and §11, we make use of the "?" gate. A d-network $\mathcal{N}$ is *coin-tossing* if and only if $\mathcal{N}$ satisfies the following conditions: (i) the d-permutation applied in the permutation phase of each level is fixed, and (ii) the type of each gate is fixed. Note that: (i) "?" gates are allowed in a coin-tossing d-network, and (ii) every deterministic d-network is a coin-tossing d-network. (We do not consider random coin-tossing networks in any of our applications. Rather, we view the "?" gate as an alternative to the form of randomness introduced above.)

A d-vector is *a-random*, $0 \le a \le d$, if and only if it is chosen from a probability distribution that assigns the same probability to any pair of d-vectors related by some d-permutation in $\Pi(d, a)$.

Let $\Pi_R(d)$ and $\Pi_R(d, a)$ denote the uniform distributions over $\Pi(d)$ and $\Pi(d, a)$, respectively.

Let $\Phi_R(d, k)$ denote the uniform distribution over $\Phi(d, k)$.

For all p in $[0, 1]$, let $\Phi_B(d, p)$ denote the distribution that assigns probability

$$p^k \cdot (1-p)^{\mathrm{pow}(d)-k}$$

to each 0-1 d-vector in $\Phi(d, k)$. Note that a random 0-1 d-vector drawn from this distribution corresponds to the sequence of outcomes of d independent, p-biased Bernoulli trials.

If $\mathcal{D}$ (resp., $\mathcal{D}'$) is the probability distribution over $\Phi(d)$ that assigns probability p_i (resp., p'_i) to the d-bit binary string $i_{d-1} \cdots i_0$, $0 \le i < \mathrm{pow}(d) = n$, then define $\mathcal{D} \le \mathcal{D}'$ if and only if there exist real numbers x_{ij} in $[0, 1]$, $0 \le i < n$, $0 \le j < n$, such that:

(i) $\sum_{0 \le i < n} x_{ij} = 1$, $0 \le j < n$,
(ii) $p'_i = \sum_{0 \le j < n} x_{ij} \cdot p_j$, $0 \le i < n$, and
(iii) $x_{ij} > 0$ only if $i_k \ge j_k$, $0 \le k < d$.

Note that Conditions (i) and (ii) ensure that $\sum_{0 \le i < n} p_i = \sum_{0 \le i < n} p'_i = 1$. Informally, $\mathcal{D} \le \mathcal{D}'$ if and only if it is possible to sample from $\mathcal{D}$ by first sampling from $\mathcal{D}'$ and then changing (according to a probability distribution that may depend on the particular sample chosen from $\mathcal{D}'$) a randomly chosen subset of the 1's to 0's.

3.7. Network Composition. The main goal of this paper is to provide efficient (i.e., small-depth) constructions of d-networks having certain sorting-related properties. In simple cases, we present our constructions by explicitly specifying the permutation phase and operation phase of each level of the d-network. However, this approach would be too cumbersome for some of our more complicated constructions.

In general, we present a given d-network as the "composition" of a sequence of: (i) explicitly specified d-networks, (ii) explicitly specified d-permutations, and (iii) recursively specified d-networks. The following mechanical procedure can be used to convert such a sequence into a d-network.

1. "Unwind" the recurrence to obtain a sequence of explicitly specified d-networks and d-permutations.
2. Repeatedly apply the composition rules specified below to adjacent pairs in the sequence until the sequence has been reduced to either: (i) a single d-network $\mathcal{N}$, or (ii) a d-network $\mathcal{N}$ followed by a d-permutation π. (The composition rules are easily seen to be associative; hence, the order of application is immaterial.) In Case (i), $\mathcal{N}$ is the desired d-network. In Case (ii), the desired d-network $\mathcal{N}'$ is obtained by appending a single level to $\mathcal{N}$, where: (i) the permutation π is applied in the permutation phase, and (ii) the operation phase consists of $\mathrm{pow}(d-1)$ "0" gates.

We now specify the three composition rules (network-network, permutation-network, and permutation-permutation) that can be applied in Step 2 above.

1. Let $\mathcal{N}$ and $\mathcal{N}'$ denote d-networks of depth a and b, respectively. Then the composition of the pair $(\mathcal{N}, \mathcal{N}')$ is the depth-$(a+b)$ d-network $\mathcal{N}''$ such that: (i) the first a levels of $\mathcal{N}''$ are given by $\mathcal{N}$, and (ii) the last b levels of $\mathcal{N}''$ are given by $\mathcal{N}'$.
2. For any depth-a d-network $\mathcal{N}$ and d-permutation π, the composition of the pair $(\pi, \mathcal{N})$ is the depth-a d-network $\mathcal{N}'$ obtained from $\mathcal{N}$ by replacing the permutation π' applied in the permutation phase of level 0 with the permutation $\pi \circ \pi'$.
3. For all d-permutations π and π', the composition of the pair (π, π') is $\pi \circ \pi'$.

Our recursive d-network constructions employ recursion over a-cubes, for some a such that $0 \le a \le d$. To achieve efficient performance, it is desirable for the the depth of a recursive construction over a-cubes be a function of a, and not d. The following definitions are extremely helpful for establishing results of this kind.

A d-network $\mathcal{N}$ is *a-partitionable*, $0 \le a \le d$, if and only if $\mathcal{N}$ can be partitioned into $\mathrm{pow}(d-a)$ disjoint a-networks $\mathcal{N}_i$, $0 \le i < \mathrm{pow}(d-a)$, where the input (resp., output) wires of $\mathcal{N}_i$ correspond to the ith input (resp., output) a-cube of $\mathcal{N}$.

Let $\mathcal{N}$ denote an a-partitionable d-network, and define $\mathcal{N}_i$ as in the preceding paragraph, $0 \le i < \mathrm{pow}(d-a)$. Then $\mathcal{N}$ is a *(d,a)-network*, $0 \le a \le d$, if and only if the $\mathcal{N}_i$'s are all identical.

We remark that: (i) every d-network is a (d,d)-network, (ii) the set of (d,a)-networks is closed under composition, (iii) for all (d,a)-networks $\mathcal{N}$ and d-permutations π in $\Pi(d,a)$, the composition of the pair $(\pi, \mathcal{N})$ is a (d,a)-network, and (iv) Lemma 4.10 provides a useful alternative characterization of the class of a-partitionable hypercubic d-networks.

3.8. Network Families. It is straightforward to prove by induction on the leveled structure of any d-network that the output d-vector is related to the input

d-vector by some d-permutation. We say that a d-network is a *sorting network* if and only if, in addition, every possible input d-vector leads to a sorted output d-vector. (In the case of a random d-network, a given input d-vector may not always produce the same output d-vector. We say that a given input vector is sorted by a random network $\mathcal{N}$ if and only if it is guaranteed to be sorted by $\mathcal{N}$.) As indicated in §1, a d-network $\mathcal{N}$ is a sorting network if and only if: (i) $\mathcal{N}$ sorts all d-permutations in $\Pi(d)$, or (ii) $\mathcal{N}$ sorts all 0-1 d-vectors in $\Phi(d)$. Because of these facts, we will often find it useful to restrict our attention to inputs drawn from $\Pi(d)$ or $\Phi(d)$.

For any d-network $\mathcal{N}$, let $Sort(\mathcal{N})$ denote the set of all integer d-vectors sorted by $\mathcal{N}$.

For any (d, a)-network $\mathcal{N}$, let $Sort(\mathcal{N}, a)$ denote the set of all integer input d-vectors to $\mathcal{N}$ for which every output a-cube is sorted.

The *depth-a shuffle-"+" d-network* is the depth-a hypercubic d-network in which every level consists of the d-permutation $\hookleftarrow_d$ followed by a set of $\text{pow}(d-1)$ "+" gates. We define other networks similarly; for example, each level of a depth-a unshuffle-"0" d-network consists of the d-permutation $\hookrightarrow_d$ followed by a set of $\text{pow}(d-1)$ "0" gates.

A d-network $\mathcal{N}$ is in $Sort_N(d, \varepsilon)$ if and only if the output of $\mathcal{N}$ is sorted with probability at least $1-\varepsilon$ on any d-random 0-1 input d-vector.

A (d, a)-network $\mathcal{N}$ is in $Sort_N(d, a, \varepsilon)$ if and only if each output a-cube of $\mathcal{N}$ is sorted with probability at least $1-\varepsilon$ on any a-random 0-1 input d-vector.

A (d, a)-network $\mathcal{N}$ is in $Sort_N(d, a, b, \varepsilon)$, $0 \le b \le a$, if and only if each output a-cube of $\mathcal{N}$ is b-sorted with probability at least $1-\varepsilon$ on any a-random 0-1 input d-vector.

A 0-1 d-vector ϕ is *a-mostly-sorted* with respect to permutation π, $0 \le a \le d$, if and only if after applying d-permutation π, the length-$(\text{pow}(d) - \text{pow}(a))$ prefix of the resulting 0-1 d-vector is a-sorted.

A (d, a)-network $\mathcal{N}$ is in $Most_N(d, a, b, \varepsilon)$, $0 \le b \le a$, if and only if there exists a d-permutation π in $\Pi(d, a)$ such that each output a-cube of $\mathcal{N}$ is b-mostly-sorted with respect to π with probability at least $1-\varepsilon$ on any a-random 0-1 input d-vector.

For nonnegative integers a and b, an *(a, b)-merge* operation takes as input $\text{pow}(b)$ sorted lists of length $\text{pow}(a)$ and produces a single sorted list of length $\text{pow}(a+b)$.

A $(d, a+b)$-network $\mathcal{N}$ is in $Merge_N(d, a, b)$, if and only if $\mathcal{N}$ performs an (a, b)-merge operation on each $(a+b)$-cube. We assume the following input convention within each $(a+b)$-cube A: The ith sorted input list is provided in ascending order in a-cube i of A, $0 \le i < \text{pow}(b)$.

A *d-insertion* operation, $d \ge 0$, takes as input a sorted list of length $\text{pow}(d) - 1$ and one additional input, and produces a sorted list of length $\text{pow}(d)$.

A (d, a)-network $\mathcal{N}$ is in $Insert_N(d, a)$, $0 \le a \le d$, if and only if $\mathcal{N}$ performs an a-insertion operation on each a-cube. We assume the following input convention within each a-cube: The sorted list is provided in ascending order on input wires 0 through $\text{pow}(a) - 2$ of the a-cube, and the additional input is provided on input wire $\text{pow}(a) - 1$ of the a-cube.

In the preceding paragraphs, we have defined a number of network families. In each case, the name of the family is subscripted by the letter "N" (for "network"). For any particular family $\mathcal{F}_N$, we define $\mathcal{F}_D$ as the minimum depth of any network in $\mathcal{F}_N$. (For example, $Sort_D(d, a, b, \varepsilon)$ denotes the minimum depth of any network in $Sort_N(d, a, b, \varepsilon)$.) Furthermore, we let $\mathcal{F}_N^h$ denote the set of all hypercubic networks in $\mathcal{F}_N$, and $\mathcal{F}_D^h$ denote the minimum depth of any network in $\mathcal{F}_N^h$.

4. Basic Lemmas. In this section, we present a number of basic lemmas.

LEMMA 4.1. *A d-network $\mathcal{N}$ is a sorting network if and only if*

$$\Phi(d) \subseteq Sort(\mathcal{N}).$$

Proof. This lemma is known as the 0-1 principle for sorting networks, and is proven in [11, §5.3.4]. (The proof is given in the context of deterministic networks. The extension to random networks is immediate, however, since a random network $\mathcal{N}$ is a sorting network if and only if every deterministic network that is assigned a non-zero probability by the distribution associated with $\mathcal{N}$ is a sorting network.) □

LEMMA 4.2. *For any d-network $\mathcal{N}$ and d-permutation π, we have*

$$\pi \in Sort(\mathcal{N}) \iff \Phi_\pi \subseteq Sort(\mathcal{N}).$$

Proof. This result follows from a slight modification to the proof of the 0-1 principle cited above. □

LEMMA 4.3. *For all a and d such that $0 < a \leq d$, we have*

$$\begin{aligned} Merge_D(d, a-1, 1) &\leq a, \\ Merge_D^h(d, a-1, 1) &= O(a). \end{aligned}$$

Proof. These bounds are established by Batcher's bitonic merge network [4]. For a hypercubic construction, additional depth is required in order to conform with the input and output conventions adopted in §3. (Our input and output conventions have been chosen in order to simplify the presentation, and not to minimize the constant factors associated with our hypercubic constructions.) This is accomplished by preceding Batcher's bitonic merge network with an appropriate fixed permutation π. (Note that such a fixed permutation does not contribute to the depth of a non-hypercubic construction.)

The role of the fixed permutation π is twofold: (i) to reverse one of the two sorted input lists within each a-cube, as required by Batcher's bitonic merge, and (ii) to "compensate" for the series of a shuffle permutations accompanying the merge (as described below). It is straightforward to implement an appropriate permutation π with an $O(a)$-depth hypercubic d-network. (We could use Lemma 4.7 for this purpose, although the permutation π is simple enough to implement directly.) A depth-a shuffle-"+" d-network can then be used to implement Batcher's bitonic merge network within each a-cube. □

LEMMA 4.4. *For all a and d such that $0 \leq a \leq d$, we have*

$$\begin{aligned} Insert_D(d, a) &\leq a, \\ Insert_D^h(d, a) &= O(a). \end{aligned}$$

Proof. This bound is also established by Batcher's bitonic merge network [4]. In contrast with the construction of Lemma 4.3, no list reversal is required since the input is already in bitonic form. (We remark that in the classic sorting network model, where a given level may contain fewer than pow$(d-1)$ gates, it is possible to match this depth bound while achieving size $\mathrm{pow}(d)-1$ instead of $d \cdot \mathrm{pow}(d)$ [14] (see also [12, §3.5.4]). The basic idea is to use a tree-like network.) □

LEMMA 4.5. *For all a and d such that $0 \leq a \leq d$, we have*

$$Sort_D^h(d, a, 0) = O(a^2).$$

Proof. This bound is due to Batcher [4], and follows from repeated application of Lemma 4.3. □

LEMMA 4.6. *For all a and d such that $0 \leq a \leq d$, we have*

$$Sort_D(d, a, 0) = O(a).$$

Proof. This bound is due to Ajtai, Komlós and Szemerédi [2]. Unfortunately, the constant factor associated with the AKS sorting network is impractically large. □

LEMMA 4.7. *For each d-permutation π in $\Pi(d, a)$, there is a hypercubic (d, a)-network $\mathcal{N}$ with the following properties: (i) $\mathcal{N}$ implements the permutation π, (ii) $\mathcal{N}$ has depth exactly $2 \cdot a$, (iii) the d-permutation $\hookrightarrow_d$ is applied in the permutation phase of each of the first a levels of $\mathcal{N}$, (iv) the d-permutation $\hookleftarrow_d$ is applied in the permutation phase of each of the last a levels of $\mathcal{N}$, and (v) every gate of $\mathcal{N}$ is either "0" or "1". Furthermore, the gate assignments of $\mathcal{N}$ can be computed in time polynomial in* $\mathrm{pow}(a)$.

Proof. This is a straightforward consequence of the work of Beneš [5]. In particular, for $a = d$, the Beneš permutation network corresponds to a hypercubic d-network satisfying properties (i), (iii), and (v). Furthermore, properties (ii) and (iv) are very nearly satisfied by the same construction; the d-network has depth $2 \cdot d - 1$ and applies the d-permutation $\hookleftarrow_d$ in the last $d - 1$ levels. It follows trivially that the claim of the lemma holds for $a = d$. (We can simply append a dummy shuffle level to the depth-$(2 \cdot d - 1)$ Beneš permutation network corresponding to an unshuffled version of π.)

For implementing a permutation in $\Pi(d, a)$, we apply our modified Beneš construction to each a-cube via the corresponding (d, a)-network of depth $2 \cdot a$. (The original Beneš construction could not be used in this manner; for $0 < a < d$, it would not map input a-cubes to output a-cubes.) □

LEMMA 4.8. *Let $\mathcal{D}$ denote an arbitrary probability distribution over the set of all d-vectors, and $\mathcal{N}$ denote a random coin-tossing depth-a d-network. If $\mathcal{N}$ sorts a random d-vector drawn from $\mathcal{D}$ with probability at least p, then there exists some deterministic depth-a d-network $\mathcal{N}'$ with the same property.*

Proof. A simple averaging argument. (Note that $\mathcal{N}$ is drawn from some fixed probability distribution over the set of all deterministic depth-a d-networks.) □

LEMMA 4.9. *Let random 0-1 d-vector ϕ be drawn from an arbitrary probability distribution over $\Phi(d)$, and π be a random d-permutation drawn from $\Pi_R(d, a)$. Then the 0-1 d-vector ϕ' obtained by applying π to ϕ is a-random.*

Proof. Straightforward. □

LEMMA 4.10. *Let $\mathcal{N}$ denote a depth-b hypercubic d-network, and for each i, $0 \leq i \leq b$, let $f^+(i)$ (resp., $f^-(i)$) denote the number of levels j of $\mathcal{N}$, $0 \leq j < i$, such that $\hookrightarrow_d$ (resp., $\hookleftarrow_d$) is applied in the permutation phase. Let $f(i) = f^+(i) - f^-(i)$, $0 \leq i \leq b$. For all a such that $0 \leq a < d$, $\mathcal{N}$ is a-partitionable if and only if: (i) $0 \leq f(i) \leq a$, $0 \leq i < b$, and (ii) $f(b) = 0$.*

Proof. Let $g^+(i) = \max_{0 \le j \le i} f(j)$ and $g^-(i) = \min_{0 \le j \le i} f(j)$, $0 \le i \le b$. Note that $g^+(i) \ge 0$ and $g^-(i) \le 0$, $0 \le i \le b$.

Let $A_{i,j}$ denote the set of level i output wires y of $\mathcal{N}$ such that there is a path to y from some input wire x in the jth input a-cube, $0 \le i < b$, $0 \le j < \mathrm{pow}(d-a)$.

It is straightforward to prove by induction on i that

$$A_{i,j} = \{y : j_{k-a} = y_{k-f(i)},\ \max\{a, g^+(i)\} \le k < d - g^-(i)\}. \tag{1}$$

Thus,

$$|A_{i,j}| = \mathrm{pow}(\min\{d, \max\{a, g^+(i)\} - g^-(i)\}). \tag{2}$$

Note that if $|A_{i,j}| > \mathrm{pow}(a)$ for some i and j, $0 \le i < b$, $0 \le j < \mathrm{pow}(d-a)$, then $\mathcal{N}$ is not a-partitionable. It follows from Equation (2) that $\mathcal{N}$ is not a-partitionable if $g^+(b) > a$ or $g^-(b) < 0$.

It remains to prove that if $g^-(b) = 0$ and $g^+(b) \le a$, then $\mathcal{N}$ is a-partitionable if and only if $f(b) = 0$. Accordingly, assume that $g^-(b) = 0$ and $g^+(b) \le a$. By Equation (1), $A_{b,j}$ corresponds to the jth output subcube if and only of $f(b) = 0$, $0 \le j < \mathrm{pow}(d-a)$, completing the proof. □

LEMMA 4.11. *Let $\mathcal{N}$ be a random coin-tossing depth-a d-network, and p (resp., p') denote the probability that a particular output wire x receives a 0 when the input to $\mathcal{N}$ is a d-vector drawn from the probability distribution $\mathcal{D}$ (resp., $\mathcal{D}'$). If $\mathcal{D} \le \mathcal{D}'$ then $p \ge p'$.*

Proof. If $\mathcal{N}$ is deterministic, the claim follows easily from consideration of the following "monotone" property of deterministic networks: When the value passed to a single input wire is changed from 0 to 1 (resp., from 1 to 0), no output changes from 1 to 0 (resp., 0 to 1).

If $\mathcal{N}$ is not deterministic, then it is given by some fixed probability distribution $\mathcal{D}''$ over the set of all deterministic depth-a d-networks. Since the claim holds for every deterministic network, we can prove that the claim holds for $\mathcal{N}$ by averaging over $\mathcal{D}''$. □

5. Analysis of the No-Elimination Tournament. Let us define a *0-1 no-elimination (d,p)-tournament*, $d \ge 0$, as an execution of the depth-d shuffle-"+" d-network on a random 0-1 input d-vector drawn from the distribution $\Phi_B(d,p)$. In this section, we analyze the behavior of the 0-1 no-elimination tournament. Our analysis culminates with Lemma 5.16, which establishes that the 0-1 no-elimination tournament has a surprisingly strong ranking property. This ranking property is used to carry out the applications of subsequent sections. (It is noteworthy that the depth-d shuffle-"+" d-network is equivalent to Batcher's bitonic merge network. We have chosen not to adopt Batcher's terminology because we plan to expose a property of the network that is largely unrelated to merging.)

The central idea underlying the proof of Lemma 5.16 is that for almost all output wires x of a 0-1 no-elimination (d,p)-tournament, the probability that x receives a 0 (which is a function of p) exhibits sharp threshold behavior: There is a probability q (which depends on x) such that x is extremely unlikely (resp., likely) to receive a 0 whenever p is at least a bit smaller (resp., larger) than q. Thus, if we permute the output wires of the 0-1 no-elimination (d,p)-tournament according to the sorted order of their associated threshold probabilities, we will tend to produce an approximately-sorted 0-1 output vector.

The proof of Lemma 5.16 is rather lengthy, and has been organized into a number of sections. In §5.1, we define and analyze certain output probability polynomials.

In §5.2, we study the inverse functions associated with these output probability polynomials. In §5.3, we provide a number of auxiliary definitions. In §5.4, we present several technical lemmas. In §5.5, we complete the proof of Lemma 5.16.

5.1. The Output Polynomials. In this section, we analyze the probability that each wire in a 0-1 no-elimination (d,p)-tournament carries a 0. We define the output probability polynomials $\sigma_\alpha(p)$ and prove two basic lemmas concerning these polynomials.

LEMMA 5.1. *Let x_0 and x_1 denote the two intermediate wires associated with some gate in a 0-1 no-elimination (d,p)-tournament. Then the events $E_0 =$ "x_0 receives a 0" and $E_1 =$ "x_1 receives a 0" are independent.*

Proof. Assume without loss of generality that: (i) x_0 is intermediate wire $2\cdot j$ on level i, and (ii) x_1 is intermediate wire $2\cdot j+1$ on level i, $0 \le i < d$, $0 \le j < \text{pow}(d-1)$. Note that the index of every level 0 input wire with a path to x_0 has a 0 in bit position $d-i-1$. Similarly, the index of every level 0 input wire with a path to x_1 has a 1 in bit position $d-i-1$. Thus, wires x_0 and x_1 depend on disjoint subsets of the level 0 input wires, and the claim of the lemma follows. ∎

With each binary string α, we associate the function $\sigma_\alpha(p)$, defined inductively as follows:

(i) $\sigma_\epsilon(p) = p$,
(ii) $\sigma_{\alpha 0}(p) = 2\cdot\sigma_\alpha(p) - \sigma_\alpha(p)^2$, and
(iii) $\sigma_{\alpha 1}(p) = \sigma_\alpha(p)^2$.

One may easily verify that for each binary string α, the following conditions hold: (i) $\sigma_\alpha(0) = 0$, (ii) $\sigma_\alpha(1) = 1$, (iii) $\sigma_\alpha(p)$ is monotonically increasing for p in $[0,1]$, (iv) $\sigma_\alpha(p)$ is a degree-$\text{pow}(|\alpha|)$ polynomial in p. Conditions (i), (ii), and (iii) imply that $\sigma_\alpha(p)$ is in $[0,1]$ for all p in $[0,1]$.

LEMMA 5.2. *Output wire j of a 0-1 no-elimination (d,p)-tournament receives a 0 with probability $\sigma_\alpha(p)$, where $\alpha = j_{d-1}\cdots j_0$.*

Proof. We prove instead the following stronger claim, for all i and j such that $0 \le i < d$, $0 \le j < \text{pow}(d)$:

(i) Input wire j at level i receives a 0 with probability $\sigma_\alpha(p)$, where $\alpha = j_{i-1}\cdots j_0$.
(ii) Intermediate wire j at level i receives a 0 with probability $\sigma_\alpha(p)$, where $\alpha = j_i \cdots j_1$.
(iii) Output wire j at level i receives a 0 with probability $\sigma_\alpha(p)$, where $\alpha = j_i\cdots j_0$.

For $i = 0$, Part (i) of the claim is immediate since $\sigma_\epsilon(p) = p$. Now let us assume that Part (i) of the claim holds for some i, $0 \le i < d$. Then Part (ii) of the claim holds for i since the value received by input wire j is passed to intermediate wire $j_{d-2}\cdots j_0 j_{d-1}$. It is similarly easy to show that if Part (iii) of the claim holds for some i, $0 \le i < d-1$, then Part (i) holds for $i+1$, since output wire j at level i is the same as input wire j at level $i+1$.

It remains only to prove that if Part (ii) of the claim holds for some i, $0 \le i < d$, then Part (iii) holds for i. Accordingly, let x_0 and x_1 denote the pair of intermediate wires associated with some gate y at level i, assume that Part (ii) of the claim holds for these wires, and that the associated d-bit indices of x_0 and x_1 are $\beta\alpha 0$ and $\beta\alpha 1$, respectively, where $|\alpha| = i$. Then

$$\Pr\{x_0 \text{ receives a } 0\} = \Pr\{x_1 \text{ receives a } 0\} = \sigma_\alpha(p),$$

and these probabilities are independent by Lemma 5.1. Hence, the "min" output of gate y (i.e., the output wire with index $\beta\alpha 0$ at level i) receives a 0 with probability

$$2\cdot\sigma_\alpha(p) - \sigma_\alpha(p)^2 = \sigma_{\alpha 0}(p),$$

and the "max" output of gate y (i.e., the output wire with index $\beta\alpha 1$ at level i) receives a 0 with probability

$$\sigma_\alpha(p)^2 = \sigma_{\alpha 1}(p),$$

as required. □

Let α and β denote the binary sequences corresponding to the win-loss sequences `WLWLLWLL` and `LLLWWWWW` mentioned in §1. We can easily calculate that $\sigma_\alpha(1/2) \approx 0.796$ and $\sigma_\beta(1/2) \approx 0.882$, suggesting that the player with record α should be rated above the player with record β.

LEMMA 5.3. *For all binary strings α and β, and all p in $[0,1]$,*

$$\sigma_{\beta\alpha}(p) = \sigma_\alpha(\sigma_\beta(p)).$$

Proof. For $\alpha = \epsilon$, the result is immediate since $\sigma_\epsilon(p) = p$. For $|\alpha| > 0$, we prove the result by induction on $|\alpha|$. For the base case, assume that $\alpha = x$, where x is either 0 or 1. Since $\sigma_0(p) = 2 \cdot p - p^2$ and $\sigma_1(p) = p^2$, we find that $\sigma_{\beta x}(p) = \sigma_x(\sigma_\beta(p))$, as required. Our induction hypothesis is that the claim holds for all α and β with $|\alpha| \leq i$, for some $i \geq 1$. For the induction step, we will prove that the claim holds for all α, β with $\alpha = \alpha' x$, x equal to 0 or 1, and $|\alpha'| = i$. The proof follows from three applications of the induction hypothesis, since

$$\sigma_{\beta\alpha}(p) = \sigma_{\beta\alpha' x}(p) = \sigma_x(\sigma_{\beta\alpha'}(p)) = \sigma_x(\sigma_{\alpha'}(\sigma_\beta(p))) = \sigma_{\alpha' x}(\sigma_\beta(p)) = \sigma_\alpha(\sigma_\beta(p)).$$

□

5.2. The Inverses of the Output Polynomials. In order to better understand the behavior of the output polynomial σ_α, it will be useful to study its inverse function. In particular, for any binary string α, we define $\Gamma_\alpha(z)$ to be the function such that

$$\Gamma_\alpha(\sigma_\alpha(p)) = p$$

for all p in $[0,1]$. Unlike σ_α, Γ_α is not a polynomial for $|\alpha| \geq 1$. However, like σ_α, there is a simple inductive scheme for computing Γ_α. This is demonstrated by the following lemma.

LEMMA 5.4. *For all binary strings α, and all z in $[0,1]$,*

$$\begin{aligned}
\Gamma_\epsilon(z) &= z, \\
\Gamma_{0\alpha}(z) &= 1 - \sqrt{1 - \Gamma_\alpha(z)}, \text{ and} \\
\Gamma_{1\alpha}(z) &= \sqrt{\Gamma_\alpha(z)}.
\end{aligned}$$

Proof. Since $\sigma_\epsilon(p) = p$ for all p in $[0,1]$, σ_ϵ is the identity function, and thus Γ_ϵ is also the identity function. Hence $\Gamma_\epsilon(z) = z$ for all z in $[0,1]$.

By Lemma 5.3, we have

$$\begin{aligned}
\sigma_{0\alpha}(p) &= \sigma_\alpha(\sigma_0(p)) \\
&= \sigma_\alpha(2 \cdot p - p^2)
\end{aligned}$$

for all p in $[0,1]$. Setting $p = \Gamma_{0\alpha}(z)$, we find that

$$\begin{aligned} \sigma_\alpha(2 \cdot \Gamma_{0\alpha}(z) - \Gamma_{0\alpha}(z)^2) &= \sigma_{0\alpha}(\Gamma_{0\alpha}(z)) \\ &= z \\ &= \sigma_\alpha(\Gamma_\alpha(z)). \end{aligned}$$

Since σ_α is a monotonically increasing function, we have

$$2 \cdot \Gamma_{0\alpha}(z) - \Gamma_{0\alpha}(z)^2 = \Gamma_\alpha(z).$$

Solving for $\Gamma_{0\alpha}(z)$, we obtain

$$\Gamma_{0\alpha}(z) = 1 - \sqrt{1 - \Gamma_\alpha(z)},$$

as desired.

The proof that $\Gamma_{1\alpha}(z) = \sqrt{\Gamma_\alpha(z)}$ proceeds in a similar fashion. By Lemma 5.3, we have

$$\begin{aligned} \sigma_{1\alpha}(p) &= \sigma_\alpha(\sigma_1(p)) \\ &= \sigma_\alpha(p^2) \end{aligned}$$

for all p in $[0,1]$. Setting $p = \Gamma_{1\alpha}(z)$, we find that

$$\begin{aligned} \sigma_\alpha(\Gamma_{1\alpha}(z)^2) &= \sigma_{1\alpha}(\Gamma_{1\alpha}(z)) \\ &= z \\ &= \sigma_\alpha(\Gamma_\alpha(z)). \end{aligned}$$

Since σ_α is a monotonically increasing function, we have

$$\Gamma_{1\alpha}(z)^2 = \Gamma_\alpha(z)$$

and thus

$$\Gamma_{1\alpha}(z) = \sqrt{\Gamma_\alpha(z)},$$

as desired. ∎

Let α and β denote the binary sequences corresponding to the win-loss sequences `WLWLLWLL` and `LLLWWWWW` mentioned in §1. We can easily calculate that $\Gamma_\alpha(1/2) \approx 0.437$ and $\Gamma_\beta(1/2) \approx 0.381$, suggesting that the player with record α should be rated above the player with record β.

Note that $\Gamma_\alpha(0) = 0$ and $\Gamma_\alpha(1) = 1$ for all binary strings α. The following lemma is analogous to Lemma 5.3.

LEMMA 5.5. *For all binary strings α and β, and all z in* $[0,1]$,

$$\Gamma_{\beta\alpha}(z) = \Gamma_\beta(\Gamma_\alpha(z)).$$

Proof. Similar to the proof of Lemma 5.3. ∎

5.3. Auxiliary Definitions. In this section, we state a number of definitions related to the analysis of the no-elimination tournament. These definitions are used primarily in §5.4 and §5.5, but also appear in subsequent sections.

For all $x < y$ in $[0, 1]$, $\lambda \geq 1$, and $d \geq 0$, let

$$\Delta(x, y) = \lg \frac{y \cdot (1 - x)}{(1 - y) \cdot x}, \tag{3}$$

$$h_\alpha(x, y) = \frac{\Delta(\Gamma_\alpha(x), \Gamma_\alpha(y))}{\Delta(x, y)}, \tag{4}$$

$$H_\lambda(x, y, d) = \sum_{\alpha : |\alpha| = d} h_\alpha(x, y)^\lambda, \tag{5}$$

$$\eta(\lambda) = \sup_{0<x<y<1} \left\{ h_0(x, y)^\lambda + h_1(x, y)^\lambda \right\}, \tag{6}$$

$$\gamma_c = \inf_{\lambda \geq 1} \frac{\lg \eta(\lambda) + \lambda}{\lambda + 1} \approx 0.822, \tag{7}$$

$$\mu_c = \sup_{\lambda \geq 1} \frac{1 - \lg \eta(\lambda)}{\lambda} = \lg(4 - 2 \cdot \sqrt{2}) \approx 0.228, \text{ and} \tag{8}$$

$$\upsilon_c = -2 \cdot \lg \mu_c = -2 \cdot \lg \lg(4 - 2 \cdot \sqrt{2}) \approx 4.260. \tag{9}$$

Informally, we think of $\Delta(x, y)$ as a measure of the "distance" between x and y for $x < y$ in $[0, 1]$. The function $h_\alpha(x, y)$ may then be viewed as the fractional decrease in the distance between x and y that results from applying Γ_α to both x and y. In §5.4, we use an inductive potential argument, with potential function $H(x, y, d)$, to show that $h_\alpha(x, y)$ is very small for most α. The function $\eta(\lambda)$ arises in the process of bounding the size of the potential function. The constants γ_c and μ_c are related to the notion of an admissible triple, which we define below. (Note that the constant γ_c and μ_c appear in the statements of Lemmas 5.6 and 5.7, respectively.) The constant υ_c appears in the exponent of the depth bound of §9.

Setting $\lambda = 3$, we can use Lemma 5.11 and elementary calculus to show that

$$\eta(3) = \frac{10 + 7 \cdot \sqrt{2}}{16},$$

which is attainable for $z = 1/2$. This implies $\gamma_c \leq [\lg(10 + 7 \cdot \sqrt{2}) - 1]/4 \approx 0.829$ and $\mu_c \geq [5 - \lg(10 + 7 \cdot \sqrt{2})]/3 = \lg(4 - 2 \cdot \sqrt{2}) \approx 0.228$. Using numerical calculations, it can be shown that $\gamma_c \approx 0.822$, which is attained for $\lambda \approx 3.609$ and $\eta(\lambda) \approx 1.133$. On the other hand, the supremum of $(1 - \lg \eta(\lambda))/\lambda$, $\lambda \geq 1$, is actually achieved for $\lambda = 3$, so $\mu_c = \lg(4 - 2 \cdot \sqrt{2})$.

DEFINITION 5.1. *A triple (γ, ε, d), where $0 < \gamma < 1$, $0 \leq \varepsilon < 1/2$, and $d \geq 0$, is defined to be* admissible *if and only if*

$$\gamma \cdot d \cdot (\lambda + 1) \geq d \cdot \lg \eta(\lambda) + \lambda \cdot [d + \lg \lg(1/\varepsilon) + 2 - \lg(1 - 2 \cdot \varepsilon)]$$

for some $\lambda \geq 1$.

Definition 5.1 is somewhat messy to apply directly. The following pair of technical lemmas characterize two classes of admissible triples that arise in our applications.

LEMMA 5.6. *For each function $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(O(d)))$, there is a function*

$$f(d) = O(\lg \lg(1/\varepsilon(d)))/d$$

such that $(\gamma, \varepsilon(d), d)$ *is an admissible triple for all* $d \geq 0$ *and* $\gamma_c + f(d) \leq \gamma < 1$.

Proof. This follows from routine calculations involving Definition 5.1 and Equation (7). □

LEMMA 5.7. *For each function* $\varepsilon(d) = \text{pow}(-\text{pow}(\mu \cdot d))$, *where* $\mu(d) = \mu_c - \frac{1}{f(d)}$ *with* $f(d) = \omega(1)$ *and* $f(d) = o(d)$, *there is a function* $g(d) = o(1)$ *such that*

$$\left(1 - \frac{3 - g(d)}{4 \cdot f(d)}, \varepsilon(d), d\right)$$

is an admissible triple for all $d \geq 0$.

Proof. This follows from routine calculations involving Definition 5.1 and Equation (8). □

5.4. Several Technical Lemmas. In this section, we prove a number of technical lemmas that are only used in §5.5 of the paper. Lemma 5.8 shows that the notion of "distance" associated with the function $\Delta(x, y)$ is always at least twice the difference $y - x$. Lemma 5.9 provides a useful method for recursively rewriting expressions of the form $h_{\beta\alpha}(x, y)$. Lemma 5.10 gives a maximization result that is used within Lemma 5.11 to obtain a simplified definition of $\eta(\lambda)$. Lemma 5.12 proves that the potential function $H(x, y, d)$ is bounded from above by $\eta(\lambda)^d$. Lemma 5.13 shows that for certain small values of ε, the difference $\Gamma_\alpha(1 - \varepsilon) - \Gamma_\alpha(\varepsilon)$ is small for most binary strings α.

LEMMA 5.8. *For all* $x < y$ *in* $[0, 1]$,

$$y - x \leq \Delta(x, y)/2.$$

Proof. Define

$$\rho(z) = \frac{1}{4} \cdot \ln \frac{z}{1 - z} - z$$

for z in $[0, 1]$. Since

$$\frac{d\rho(z)}{dz} = \frac{1}{4} \cdot \left(\frac{1}{z} + \frac{1}{1 - z}\right) - 1 \geq 0$$

for z in $[0, 1]$, we know that $\rho(z)$ is a non-decreasing function of z. Hence,

$$\begin{aligned} \frac{\Delta(x, y)}{4 \cdot \lg e} - (y - x) &= \frac{\lg \frac{y \cdot (1-x)}{(1-y) \cdot x}}{4 \cdot \lg e} - y + x \\ &= \rho(y) - \rho(x) \\ &\geq 0, \end{aligned}$$

and thus

$$y - x \leq \frac{\Delta(x, y)}{4 \lg e} \leq \Delta(x, y)/2.$$

□

LEMMA 5.9. *For all* $x < y$ *in* $[0, 1]$, *we have: (i)* $h_\epsilon(x, y) = 1$, *and (ii) for all binary strings* α *and* β,

$$h_{\beta\alpha}(x, y) = h_\beta(\Gamma_\alpha(x), \Gamma_\alpha(y)) \cdot h_\alpha(x, y).$$

Proof. Since Γ_ϵ is the identity function, $h_\epsilon(x,y) = 1$ for all $x < y$ in $[0,1]$. We prove the second part of the lemma by observing that

$$\begin{aligned} h_{\beta\alpha}(x,y) &= \frac{\Delta(\Gamma_{\beta\alpha}(x), \Gamma_{\beta\alpha}(y))}{\Delta(x,y)} \\ &= \frac{\Delta(\Gamma_{\beta\alpha}(x), \Gamma_{\beta\alpha}(y))}{\Delta(\Gamma_\alpha(x), \Gamma_\alpha(y))} \cdot \frac{\Delta(\Gamma_\alpha(x), \Gamma_\alpha(y))}{\Delta(x,y)} \\ &= \frac{\Delta(\Gamma_\beta(\Gamma_\alpha(x)), \Gamma_\beta(\Gamma_\alpha(y)))}{\Delta(\Gamma_\alpha(x)\Gamma_\alpha(y))} \cdot h_\alpha(x,y) \\ &= h_\beta(\Gamma_\alpha(x), \Gamma_\alpha(y)) \cdot h_\alpha(x,y), \end{aligned}$$

where the second last equation follows from Lemma 5.5. □

LEMMA 5.10. *Let f_0, f_1, and f denote strictly increasing and continuously differentiable functions on $(0,1)$, and set*

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

for $0 < x \le y < 1$ and $\lambda \ge 1$. Then for all $x \le y$ in $(0,1)$,

$$g(x,y,\lambda) \le \sup_{z \in (0,1)} g(z,z,\lambda).$$

Proof. Note that because f_0, f_1, and f are strictly increasing and differentiable, l'Hôpital's rule implies that $g(x,y,\lambda)$ is well-defined even if $x = y$.

Given any $x < y$ in $(0,1)$, we prove below that there exists a p such that $x < p < y$ and

$$\max\{g(x,p,\lambda), g(p,y,\lambda)\} \ge g(x,y,\lambda).$$

This is sufficient to prove that the maximum of $g(x,y,\lambda)$ occurs for $x \sim y$.

Choose p so that

$$f(p) = \frac{f(x) + f(y)}{2}.$$

We can always find such a p between x and y since f is a continuous function. Then set

$$\begin{aligned} u_0 &= f_0(p) - f_0(x), \\ u_1 &= f_0(y) - f_0(p), \\ v_0 &= f_1(p) - f_1(x), \\ v_1 &= f_1(y) - f_1(p), \text{ and} \\ w &= f(p) - f(x) \\ &= f(y) - f(p). \end{aligned}$$

Note that u_0, u_1, v_0, v_1, and w are all strictly positive since f_0, f_1, and f are strictly increasing.

By definition,

$$\begin{aligned} g(x,p,\lambda) &= \left(\frac{u_0}{w}\right)^\lambda + \left(\frac{v_0}{w}\right)^\lambda, \\ g(p,y,\lambda) &= \left(\frac{u_1}{w}\right)^\lambda + \left(\frac{v_1}{w}\right)^\lambda, \text{ and} \\ g(x,y,\lambda) &= \left(\frac{u_0+u_1}{2\cdot w}\right)^\lambda + \left(\frac{v_0+v_1}{2\cdot w}\right)^\lambda. \end{aligned}$$

For $\lambda \geq 1$, the function z^λ is convex, and thus

$$\frac{z_0^\lambda + z_1^\lambda}{2} \geq \left(\frac{z_0+z_1}{2}\right)^\lambda$$

for all z_0 and z_1. Hence

$$\begin{aligned} \left(\frac{u_0}{w}\right)^\lambda + \left(\frac{u_1}{w}\right)^\lambda &\geq 2\cdot\left(\frac{u_0+u_1}{2\cdot w}\right)^\lambda \text{ and} \\ \left(\frac{v_0}{w}\right)^\lambda + \left(\frac{v_1}{w}\right)^\lambda &\geq 2\cdot\left(\frac{v_0+v_1}{2\cdot w}\right)^\lambda. \end{aligned}$$

Summing the preceding pair of inequalities, we find that

$$g(x,p,\lambda) + g(p,y,\lambda) \geq 2\cdot g(x,y,\lambda),$$

and hence $\max\{g(x,p,\lambda), g(p,y,\lambda)\} \geq g(x,y,\lambda)$, as desired. ☐

LEMMA 5.11. *For all $\lambda \geq 1$,*

$$\eta(\lambda) = \sup_{0\leq z\leq 1}\left[\left(\frac{1+\sqrt{z}}{2}\right)^\lambda + \left(\frac{1+\sqrt{1-z}}{2}\right)^\lambda\right].$$

Proof. We have

$$\begin{aligned} &h_0(x,y)^\lambda + h_1(x,y)^\lambda \\ &= \left(\frac{\Delta(\Gamma_0(x),\Gamma_0(y))}{\Delta(x,y)}\right)^\lambda + \left(\frac{\Delta(\Gamma_1(x),\Gamma_1(y))}{\Delta(x,y)}\right)^\lambda \\ &= \left(\frac{\lg\frac{\Gamma_0(y)\cdot(1-\Gamma_0(x))}{(1-\Gamma_0(y))\cdot\Gamma_0(x)}}{\lg\frac{y\cdot(1-x)}{(1-y)\cdot x}}\right)^\lambda + \left(\frac{\lg\frac{\Gamma_1(y)\cdot(1-\Gamma_1(x))}{(1-\Gamma_1(y))\cdot\Gamma_1(x)}}{\lg\frac{y\cdot(1-x)}{(1-y)\cdot x}}\right)^\lambda \\ &= \left(\frac{\lg\frac{\Gamma_0(y)}{1-\Gamma_0(y)} - \lg\frac{\Gamma_0(x)}{1-\Gamma_0(x)}}{\lg\frac{y}{1-y} - \lg\frac{x}{1-x}}\right)^\lambda + \left(\frac{\lg\frac{\Gamma_1(y)}{1-\Gamma_1(y)} - \lg\frac{\Gamma_1(x)}{1-\Gamma_1(x)}}{\lg\frac{y}{1-y} - \lg\frac{x}{1-x}}\right)^\lambda \\ &= \left(\frac{f_0(y)-f_0(x)}{f(y)-f(x)}\right)^\lambda + \left(\frac{f_1(y)-f_1(x)}{f(y)-f(x)}\right)^\lambda, \end{aligned}$$

where

$$f_0(z) = \lg\frac{\Gamma_0(z)}{1-\Gamma_0(z)}$$

$$
\begin{aligned}
&= \lg \frac{1-\sqrt{1-z}}{\sqrt{1-z}}, \\
f_1(z) &= \lg \frac{\Gamma_1(z)}{1-\Gamma_1(z)} \\
&= \lg \frac{\sqrt{z}}{1-\sqrt{z}}, \text{ and} \\
f(z) &= \lg \frac{z}{1-z}.
\end{aligned}
$$

It is easily verified that $f_0(z)$, $f_1(z)$, and $f(z)$ are strictly increasing and continuously differentiable in $(0,1)$. By Lemma 5.10, this means that the supremum of $h_0(x,y)^\lambda + h_1(x,y)^\lambda$ occurs for $x \sim y$. Using l'Hôpital's rule and elementary calculus, we can show that

$$
\begin{aligned}
\lim_{\varepsilon \to 0} h_0((1-\varepsilon)\cdot y, y) &= \frac{df_0(y)/dy}{df(y)/dy} \\
&= \frac{1+\sqrt{1-y}}{2}.
\end{aligned}
$$

Reasoning in a similar fashion, we can also show that

$$
\begin{aligned}
\lim_{\varepsilon \to 0} h_1((1-\varepsilon)\cdot y, y) &= \frac{df_1(y)/dy}{df(y)/dy} \\
&= \frac{1+\sqrt{y}}{2}.
\end{aligned}
$$

The proof of the lemma now follows from the definition of $\eta(\lambda)$. □

LEMMA 5.12. *For all $x < y$ in $[0,1]$, $\lambda \geq 1$, and $d \geq 0$, we have*

$$H_\lambda(x,y,d) \leq \eta(\lambda)^d.$$

Proof. For $d = 0$, the result is immediate since $H_\lambda(x,y,0) = 1$. For $d > 0$, Lemma 5.9 implies that

$$
\begin{aligned}
H_\lambda(x,y,d) &= \sum_{\alpha:|\alpha|=d-1} \left(h_{0\alpha}(x,y)^\lambda + h_{1\alpha}(x,y)^\lambda\right) \\
&= \sum_{\alpha:|\alpha|=d-1} \left(h_0(\Gamma_\alpha(x),\Gamma_\alpha(y))^\lambda + h_1(\Gamma_\alpha(x),\Gamma_\alpha(y))^\lambda\right)\cdot h_\alpha(x,y)^\lambda \\
&\leq \sum_{\alpha:|\alpha|=d-1} \eta(\lambda)\cdot h_\alpha(x,y)^\lambda \\
&= \eta(\lambda)\cdot H_\lambda(x,y,d-1).
\end{aligned}
$$

Hence, $H_\lambda(x,y,d) \leq \eta(\lambda)^d$, as required. □

LEMMA 5.13. *For any admissible triple (γ,ε,d), there are at most $\mathrm{pow}(\gamma\cdot d)$ length-d binary strings α such that*

$$\Gamma_\alpha(1-\varepsilon) - \Gamma_\alpha(\varepsilon) > (1-2\cdot\varepsilon)\cdot \mathrm{pow}((\gamma-1)\cdot d)/4.$$

Proof. By Lemma 5.8, the definition of h_α, and the definition of Δ, we have

$$\begin{aligned}\Gamma_\alpha(1-\varepsilon)-\Gamma_\alpha(\varepsilon) &\le \Delta(\Gamma_\alpha(\varepsilon),\Gamma_\alpha(1-\varepsilon))/2\\ &= h_\alpha(\varepsilon,1-\varepsilon)\cdot\Delta(\varepsilon,1-\varepsilon)/2\\ &\le h_\alpha(\varepsilon,1-\varepsilon)\cdot \lg(1/\varepsilon).\end{aligned}$$

Hence, it is sufficient to prove that at most $\mathrm{pow}(\gamma\cdot d)$ length-d binary strings α satisfy

$$h_\alpha(\varepsilon,1-\varepsilon) > \frac{(1-2\cdot\varepsilon)\cdot\mathrm{pow}((\gamma-1)\cdot d)}{4\cdot\lg(1/\varepsilon)}.$$

Suppose the latter claim were false. Then

$$\begin{aligned}H_\lambda(\varepsilon,1-\varepsilon,d) &> \mathrm{pow}(\gamma\cdot d)\cdot\left[\frac{(1-2\cdot\varepsilon)\cdot\mathrm{pow}((\gamma-1)\cdot d)}{4\cdot\lg(1/\varepsilon)}\right]^\lambda\\ &= \mathrm{pow}(\gamma\cdot d\cdot(\lambda+1)-\lambda\cdot[d+\lg\lg(1/\varepsilon)+2-\lg(1-2\cdot\varepsilon)])\\ &\ge \eta(\lambda)^d,\end{aligned}$$

which contradicts Lemma 5.12. □

5.5. The No-Elimination Tournament Theorem. In this section, we complete the proof of Lemma 5.16.

LEMMA 5.14. *For any admissible triple (γ,ε,d), there exists a set A of at least $\mathrm{pow}(d)-\mathrm{pow}(\gamma\cdot d)$ output wires of the depth-d shuffle-"+" d-network, and a fixed permutation π of A, such that for each p the set A can be partitioned into three sets B, A^-, and A^+ where:*

(i) the set of output wires B is mapped to a contiguous interval by the permutation π,

(ii) $|B|<\mathrm{pow}(\gamma\cdot d)$,

(iii) A^- (resp., A^+) is the set of all output wires in $A\setminus B$ mapped to positions lower (resp., higher) than B by π, and

(iv) after execution of a 0-1 no-elimination (d,p)-tournament, each output wire in A^- (resp., A^+) receives a 1 (resp., 0) with probability less than ε.

Proof. Let (γ,ε,d) denote a given admissible triple, and choose A to be the set (guaranteed to exist by Lemma 5.13) of at least $\mathrm{pow}(d)-\mathrm{pow}(\gamma\cdot d)$ output wires indexed by length-d binary strings α such that

$$\Gamma_\alpha(1-\varepsilon)-\Gamma_\alpha(\varepsilon)\le\delta$$

where $\delta=(1-2\cdot\varepsilon)\cdot\mathrm{pow}((\gamma-1)\cdot d)/4$. (Note that $0<\delta<1/4$ since (γ,ε,d) is an admissible triple.) We remark that, using Lemma 5.4, $\Gamma_\alpha(z)$ can be computed in $O(|\alpha|)$ arithmetic operations (counting square root as a single operation) for any z in $[0,1]$. Hence, the set A can be computed in $O(d\cdot\mathrm{pow}(d))$ operations. (This may be viewed as a relatively efficient time bound since, for example, it is linear in the size of the depth-d shuffle-"+" d-network.)

In fact, we can compute an appropriate permutation π within the same asymptotic time bound: We set π to the permutation of set A that sorts the $\Gamma_\alpha(\varepsilon)$ values in ascending order. Ties may be broken arbitrarily. It remains to prove that our choice of A and π satisfies the requirements of the lemma.

Let $p^-=\max\{0,p-\delta\}$ and $p^+=\min\{1,p+\delta\}$. (Recall that p is the 0-1 no-elimination tournament input parameter.) Let B denote the set of binary strings

α in A for which $\Gamma_\alpha(\varepsilon)$ is contained in $[p^-, p]$. Because the σ_α's are monotonically increasing, and using linearity of expectation, we have

$$\begin{aligned}
\sum_{\alpha:|\alpha|=d} |\sigma_\alpha(p^+) - \sigma_\alpha(p^-)| &= \sum_{\alpha:|\alpha|=d} \left(\sigma_\alpha(p^+) - \sigma_\alpha(p^-)\right) \\
&= \left(\sum_{\alpha:|\alpha|=d} \sigma_\alpha(p^+)\right) - \left(\sum_{\alpha:|\alpha|=d} \sigma_\alpha(p^-)\right) \\
&= (p^+ - p^-) \cdot \mathrm{pow}(d) \\
&\leq 2 \cdot \delta \cdot \mathrm{pow}(d).
\end{aligned}$$

For each α in B we have $\sigma_\alpha(p^-) \leq \varepsilon$. (If $p^- = 0$ then $\sigma_\alpha(p^-) = \sigma_\alpha(0) = 0$, and if $p^- = p - \delta$ then $\sigma_\alpha(p^-) \leq \sigma_\alpha(\Gamma_\alpha(\varepsilon)) = \varepsilon$.) Furthermore, for each α in B we have $\sigma_\alpha(p^+) \geq 1 - \varepsilon$. (If $p^+ = 1$ then $\sigma_\alpha(p^+) = \sigma_\alpha(1) = 1$, and if $p^+ = p + \delta$ then $\sigma_\alpha(p^+) \geq \sigma_\alpha(\Gamma_\alpha(1-\varepsilon)) = 1 - \varepsilon$.) Hence,

$$|\sigma_\alpha(p^+) - \sigma_\alpha(p^-)| \geq 1 - 2 \cdot \varepsilon.$$

The preceding inequalities imply that

$$\begin{aligned}
|B| &\leq 2 \cdot \delta \cdot \mathrm{pow}(d)/(1 - 2\cdot\varepsilon) \\
&< \mathrm{pow}(\gamma \cdot d).
\end{aligned}$$

Note that the set of binary strings B satisfies Conditions (i) and (ii) of the lemma. For the given choice of B, define sets A^- and A^+ to satisfy Condition (iii). It remains only to address Condition (iv).

Let α denote the binary string associated with an arbitrary output in A^-. Thus, $\Gamma_\alpha(\varepsilon) < p^-$ which implies $\Gamma_\alpha(1-\varepsilon) < p$. Hence $\sigma_\alpha(p) > 1 - \varepsilon$. (The probability that output α receives a 1 is less than ε.)

Similarly, let α denote the binary string associated with some output in A^+. Thus, $\Gamma_\alpha(\varepsilon) > p$ and hence $\sigma_\alpha(p) < \varepsilon$. (The probability that output α receives a 0 is less than ε.) □

LEMMA 5.15. *Let d, k, n, and p be such that $d \geq 0$, $n = \mathrm{pow}(d)$, $0 \leq k < n$, and $p = k/n$. Let $\mathcal{N}$ denote an arbitrary d-network, and assume that output wire x of $\mathcal{N}$ receives a 0 (resp., 1) with probability $q \leq \varepsilon$ when the input to $\mathcal{N}$ is drawn from $\Phi_B(d,p)$. Further assume that output wire x receives a 0 (resp., 1) with probability q_i when the input is drawn from $\Phi_R(d,i)$, $0 \leq i < n$. Then $q_k \leq 2 \cdot \varepsilon$.*

Proof. Informally, the lemma states that network $\mathcal{N}$ behaves similarly on inputs drawn from $\Phi_B(d,p)$ and $\Phi_R(d,k)$. Note that

$$\begin{aligned}
q &= \sum_{0 \leq i < n} \binom{n}{i} p^i (1-p)^{n-i} q_i \\
&\geq \sum_{k \leq i < n} \binom{n}{i} p^i (1-p)^{n-i} q_i \\
&\geq q_k \sum_{k \leq i < n} \binom{n}{i} p^i (1-p)^{n-i} \\
&\geq q_k/2,
\end{aligned}$$

where the last two inequalities follow from Lemma 4.11 and Theorem B.1, respectively. □

LEMMA 5.16. *For any admissible triple* (γ, ε, d)*, we have*

$$Most_D^h(d, d, \lfloor \gamma \cdot d \rfloor, O(\mathrm{pow}(d) \cdot \varepsilon)) \leq d.$$

Proof. Let $\mathcal{N}$ denote the depth-d shuffle-"+" d-network. It follows from Lemmas 5.14 and 5.15 that $\mathcal{N}$ belongs to $Most_N^h(d, d, \lfloor \gamma \cdot d \rfloor, O(\mathrm{pow}(d) \cdot \varepsilon))$, justifying the claimed inequality.

(We have opted to omit the detailed proof of the above claim, which is straightforward but tedious. An outline of the proof is as follows. First, we rewrite the proof obligation by successively expanding the definitions of $Most_N^h(d, d, \lfloor \gamma \cdot d \rfloor, O(\mathrm{pow}(d) \cdot \varepsilon))$, $\lfloor \gamma \cdot d \rfloor$-mostly-sorted, and $\lfloor \gamma \cdot d \rfloor$-sorted. Having done this, we arrive at a new proof obligation in which the input to network $\mathcal{N}$ is assumed to be drawn from $\Phi_R(d, k)$, $0 \leq k < \mathrm{pow}(d)$. Furthermore, the new proof obligation is such that it would follow immediately from Lemma 5.14 if only the input were assumed to be drawn from $\Phi_B(d, p)$ instead of $\Phi_R(d, k)$. Lemma 5.15 is then used to complete the proof by relating the behavior of network $\mathcal{N}$ on inputs drawn from $\Phi_B(d, p)$ to its behavior on inputs drawn from $\Phi_R(d, k)$.) □

LEMMA 5.17. *For any admissible triple* (γ, ε, a) *and all* d*,* $0 \leq a \leq d$*, we have*

$$\begin{aligned} Most_D(d, a, \lfloor \gamma \cdot a \rfloor, O(\mathrm{pow}(a) \cdot \varepsilon)) &\leq a, \\ Most_D^h(d, a, \lfloor \gamma \cdot a \rfloor, O(\mathrm{pow}(a) \cdot \varepsilon)) &= O(a). \end{aligned}$$

Proof. It follows easily from Lemma 5.16 that the (d, a)-network consisting of the d-permutation $\hookrightarrow_d^a$ followed by the depth-a shuffle-"+" d-network has the desired properties. The additional depth required to implement the d-permutation $\hookrightarrow_d^a$ is $\min\{a, d-a\} \leq a$ for a hypercubic construction, and 0 for a non-hypercubic construction. □

Lemma 5.16 formalizes a central claim of the paper, namely, that the 0-1 no-elimination tournament has a surprisingly strong ranking property. Though stated in the 0-1 domain, Lemma 5.16 can easily be interpreted in the permutation domain. Such an interpretation is provided by the following theorem, which is stated without proof. (We remark that the extra factor of $\mathrm{pow}(d)$ in the error bound arises because $|\Phi_\pi| = \mathrm{pow}(d) + 1$ for any d-permutation π. Also, Lemma 5.17, and not Theorem 5.1, is used to derive the results of subsequent sections.)

THEOREM 5.1. *Let* (γ, ε, d) *be an admissible triple, and define set* A *and permutation* π *as in the proof of Lemma 5.14. Let a random* d*-permutation drawn from* $\Pi_R(d)$ *be input to a depth-*d *shuffle-"+"* d*-network, and let* π' *denote the permutation induced on the output wires of* A*. Then the permutation obtained by applying* π *to* π' *is sorted to within* $\mathrm{pow}(\gamma \cdot d)$ *positions with probability at least* $1 - O(\mathrm{pow}(2 \cdot d) \cdot \varepsilon)$*.* □

Finally, we remark that the factors of $\mathrm{pow}(d)$, $\mathrm{pow}(a)$, and $\mathrm{pow}(2 \cdot d)$ appearing in the error bounds associated with Lemma 5.16, Lemma 5.17, and Theorem 5.1 are not best possible. Lowering these factors would require a much more careful analysis, however, and from a theoretical point of view, would yield essentially no improvement to the results of subsequent sections. (Our applications make use of Lemma 5.17 with ε set far smaller than $\mathrm{pow}(-c \cdot d)$ for any constant $c > 0$.) Of course, from a practical standpoint, it would be interesting to pin down the error bounds more accurately. For sufficiently small values of d, we remark that a computer program can be used to obtain very accurate error estimates.

6. A Small-Constant-Factor Network that Sorts Most Inputs. Given Lemma 5.16, it is now a relatively simple task to design a logarithmic-depth network that sorts a random 0-1 input vector with high probability. The transformation consists of two basic components, informally outlined below:

1. A procedure for augmenting the network of Lemma 5.16, which guarantees that all but a small fixed subset of the outputs are approximately sorted on a random 0-1 input vector with high probability, to obtain a network which guarantees that the entire output vector is approximately sorted on a random 0-1 input vector with high probability.
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 of Lemma 5.16 worked on all 0-1 input vectors, and if we didn't care about constant factors, then it would be straightforward to devise a logarithmic-depth sorting network 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 proof will be somewhat more tedious. Nevertheless, we will still follow the basic approach outlined above in order to establish Lemma 6.1, the main technical result of this section.

LEMMA 6.1. *For all a and d such that $0 \le a \le d$, and each function $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(o(d)))$, there is a function $f(d) = o(1)$ such that*

$$\mathit{Sort}_D(d, a, \varepsilon(a)) \le \frac{2-\gamma_c^2+f(a)}{1-\gamma_c} \cdot a.$$

Furthermore, there is a deterministic d-network that achieves this bound. □

Note that $(2-\gamma_c^2)/(1-\gamma_c) \approx 7.44$. The following theorem provides an interpretation of Lemma 6.1 in the permutation domain.

THEOREM 6.1. *For each function $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(o(d)))$, there is a function $f(d) = o(1)$ and a deterministic d-network of depth*

$$\frac{2-\gamma_c^2+f(d)}{1-\gamma_c} \cdot d$$

that sorts a random d-permutation drawn from $\Pi_R(d)$ with probability at least $1-\varepsilon(d)$.

Proof. Let $n = \mathrm{pow}(d)$. If π is a random d-permutation drawn from $\Pi_R(d)$, then ϕ_π^k is drawn at random from $\Phi_R(d,k)$, $0 \le k \le n$. By Lemma 6.1 (with $a = d$), there is a deterministic d-network $\mathcal{N}$ of the desired depth that sorts a random 0-1 d-vector drawn from $\Phi_R(d,k)$, $0 \le k \le n$, with probability at least $1-\varepsilon(d)$ for each function $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(g(d)))/(n+1)$ with $g(d) = o(d)$. By Lemma 4.2, d-network $\mathcal{N}$ sorts d-permutation π if and only if it sorts the $n+1$ 0-1 d-vectors in Φ_π. This occurs with probability at least $1-(n+1)\cdot\varepsilon(d) = 1-\mathrm{pow}(-\mathrm{pow}(g(d)))$, as required. □

LEMMA 6.2. *For all a, b, and d such that $0 \le b \le a \le d$, and all ε and ε' in $[0,1]$, we have*

$$\mathit{Sort}_D(d, a, \varepsilon + 2\cdot\varepsilon') \le \mathit{Sort}_D(d, a, b, \varepsilon) + \mathit{Sort}_D(d, b, \varepsilon') + 2\cdot \mathit{Merge}_D(d, b, 1).$$

Proof. We may assume that $a > b$, since the claim is trivial otherwise. We argue that a (d, a)-network in $\mathit{Sort}_N(d, a, \varepsilon + 2 \cdot \varepsilon')$ can be constructed by composing: (a) any (d, a)-network in $\mathit{Sort}_N(d, a, b, \varepsilon)$, (b) a random d-permutation drawn from $\Pi_R(d, b)$, (c) any (d, a)-network in $\mathit{Sort}_N(d, b, \varepsilon')$, (d) any $(d, b + 1)$-network in $\mathit{Merge}_N(d, b, 1)$, (e) the d-permutation π in $\Pi(d, a)$ that maps wire i to wire $(i + \mathrm{pow}(b)) \bmod \mathrm{pow}(a)$, $0 \le i < \mathrm{pow}(a)$, within each a-cube, (f) any $(d, b + 1)$-network in $\mathit{Merge}_N(d, b, 1)$, (g) the d-permutation π^{-1}, and (h) the d-permutation in $\Pi(d, a)$ that exchanges the lowest and highest b-cubes within each a-cube. (Note that this construction does indeed give a (d, a)-network.)

We may assume that the input is an a-random 0-1 d-vector. Consider an arbitrary a-cube A. After stage (a) of the construction, A is b-sorted with probability at least $1 - \varepsilon$. The output of stage (b) is b-random by Lemma 4.9. Hence, each b-cube of A is sorted with probability at least $1 - \varepsilon'$ after stage (c). Furthermore, with probability at least $1 - \varepsilon$, no two non-consecutive b-cubes of A receive non-trivial input.

In what follows, we complete the proof by showing that A is sorted after stage (h) whenever: (i) A is b-sorted after stage (a), and (ii) every b-cube of A is sorted after stage (c). (Note that these conditions are satisfied with probability at least $1-\varepsilon-2\cdot\varepsilon'$.)

Accordingly, assume that conditions (i) and (ii) hold. Then A is sorted after stage (c) with the exception of at most two non-trivial b-cubes of A. If A contains 0 or 1 non-trivial b-cubes after stage (c), then A is easily seen to be sorted after stages (c), (d), and (h). If there are 2 non-trivial b-cubes in A after stage (c) then they are adjacent. If b-cubes $2 \cdot j$ and $2 \cdot j + 1$ are non-trivial for some integer j, $0 \le j < \mathrm{pow}(a - b - 1)$, then A is sorted after stages (d) and (h). If b-cubes $2 \cdot j + 1$ and $2 \cdot j + 2$ are non-trivial for some integer j, $0 \le j < \mathrm{pow}(a - b - 1) - 1$, then stage (d) has no effect and the output of stage (h) is sorted. $\square$

Lemmas 4.3, 4.6, and 6.2 together imply that

$$\mathit{Sort}_D(d, a, \varepsilon) \le \mathit{Sort}_D(d, a, b, \varepsilon) + O(b) \tag{10}$$

for all a, b, and d such that $0 \le b \le a \le d$, and all ε in $[0, 1]$.

LEMMA 6.3. *For all a, b, and d such that $0 \le b \le a \le d$, and all ε in $[0, 1]$, we have*

$$\mathit{Sort}_D(d, a, b + 1, \varepsilon) \le \mathit{Most}_D(d, a, b, \varepsilon) + \mathit{Insert}_D(d, a - b).$$

Proof. We argue that a (d, a)-network in $\mathit{Sort}_N(d, a, b + 1, \varepsilon)$ can be constructed by composing: (a) any (d, a)-network in $\mathit{Most}_N(d, a, b, \varepsilon)$, (b) an appropriate d-permutation π in $\Pi(d, a)$, (c) the d-permutation $\hookrightarrow^b_d$, (d) any $(d, a - b)$-network in $\mathit{Insert}_N(d, a - b)$, and (e) the d-permutation $\hookleftarrow^b_d$. (Note that this construction does indeed give a (d, a)-network.)

We may assume that the input is an a-random 0-1 d-vector. By the definition of $\mathit{Most}_N(d, a, b, \varepsilon)$, there is some d-permutation π in $\Pi(d, a)$ such that each a-cube is b-mostly-sorted with respect to π with probability at least $1 - \varepsilon$ after stage (a). This is the desired stage (b) d-permutation π. Consider an arbitrary a-cube A. In what follows, we complete the proof by showing that if A is b-mostly-sorted with respect to π after stage (a), then A is $(b + 1)$-sorted after stage (e).

Accordingly, let us assume that A is b-mostly-sorted with respect to π after stage (a). Then each $(a - b)$-cube of A contains an insertion instance after stage (c). Thus, each $(a - b)$-cube of A is sorted after stage (d). Furthermore, note that each

$(a-b)$-cube of A contains the same number of 0's to within 2. Hence A has a dirty region of size at most $2 \cdot \text{pow}(b) = \text{pow}(b+1)$ after stage (e), as desired. □

LEMMA 6.4. *For all a, b, and d such that $0 \le b \le a \le d$, and all γ, ε, and ε' in $[0,1]$, we have*

$$\begin{aligned} \mathit{Sort}_D(d,a,b'+3,\varepsilon+2\cdot\varepsilon') &\le \mathit{Sort}_D(d,a,b,\varepsilon) + \mathit{Sort}_D(d,b+2,b'+1,\varepsilon') \\ &\quad + \mathit{Merge}_D(d,b-b',1), \end{aligned}$$

where $b' = \lfloor \gamma \cdot (b+2) \rfloor$.

Proof. We may assume that $b > b'+3$ and $a > b+2$, since the claim is trivial otherwise. Let $m = \text{pow}(a-b-2)$. We argue that a (d,a)-network in $\mathit{Sort}_N(d,a,b'+3,\varepsilon+2\cdot\varepsilon')$ can be constructed by composing: (a) any (d,a)-network in $\mathit{Sort}_N(d,a,b,\varepsilon)$, (b) a random d-permutation drawn from $\Pi_R(d,b+2)$, (c) any $(d,b+2)$-network in $\mathit{Sort}_N(d,b+2,b'+1,\varepsilon')$, (d) an appropriate d-permutation π in $\Pi(d,a)$, (e) any $(d,b-b'+1)$-network in $\mathit{Merge}_N(d,b-b',1)$, and (f) an appropriate d-permutation π' in $\Pi(d,a)$. (Note that this construction does indeed give a (d,a)-network.)

We may assume that the input is an a-random 0-1 d-vector. By the definition of $\mathit{Sort}_N(d,a,b,\varepsilon)$, each a-cube is b-sorted with probability at least $1-\varepsilon$ after stage (a). The output of stage (b) is $(b+2)$-random by Lemma 4.9. Hence, each $(b+2)$-cube is $(b'+1)$-sorted with probability at least $1-\varepsilon'$ after stage (c). Consider an arbitrary a-cube A, and let B_i denote the ith $(b+2)$-cube of A, $0 \le i < m$.

After stage (c), note that the following claims hold with probability at least $1-\varepsilon-2\cdot\varepsilon'$: (i) the dirty region of A has size at most $\text{pow}(b)+2\cdot\text{pow}(b'+1) \le \text{pow}(b+1)$, and (ii) every B_i is $(b'+1)$-sorted. Let B_i^- (resp., B_i^+) denote $(b+1)$-cube 0 (resp., 1) of B_i. (In other words, B_i^- and B_i^+ are the "low half" and "high half" of B_i, respectively.) If condition (i) holds, then the dirty region of A is either confined to some B_i, or to some (B_{i-1}^+, B_i^-) pair, $0 < i < m$. Let us say that Case 1 holds if, after stage (c), the dirty region of A is confined to some B_i and conditions (i) and (ii) hold. Similarly, Case 2 holds if, after stage (c), the dirty region of A is confined to some (B_{i-1}^+, B_i^-) pair and conditions (i) and (ii) hold. Otherwise, Case 3 holds. Note that Case 3 holds with probability at most $\varepsilon + 2\varepsilon'$.

We now define the d-permutation π to be applied in stage (d). Break each $(b+1)$-cube B_i^+ (resp., B_i^-) into $\text{pow}(b'+1)$ equal-sized sets $B_{i,j}^+$ (resp., $B_{i,j}^-$) by applying the $(b+1)$-permutation $\hookrightarrow_{b+1}^{b'+1}$ and then partitioning into $(b-b')$-cubes. (In other words, the set $B_{i,j}^+$ consists of those wires in B_i^+ with indices congruent to j modulo $\text{pow}(b'+1)$, $0 \le j < \text{pow}(b'+1)$.) In the arguments that follow, let C_i denote $B_{i-1}^+ \cup B_i^-$, and C_i^j denote the jth $(b-b'+1)$-cube of C_i, $0 < i < m$, $0 \le j < \text{pow}(b'+1)$. The d-permutation π is defined in such a way that: (i) the wires in B_0^- and B_{m-1}^+ are left alone, and (ii) for each (B_{i-1}^+, B_i^-) pair, $0 < i < m$, the wires of $B_{i-1,j}^+$ and $B_{i,j}^-$ are brought into opposite halves of $(b-b'+1)$-cube C_i^j in preparation for the merge step of stage (e), $0 \le j < \text{pow}(b'+1)$.

If either Case 1 or Case 2 holds after stage (c), note that the following claims hold after stage (d), $0 < i < m$: (i) all of the $B_{i-1,j}^+$'s and $B_{i,j}^-$'s are sorted, and (ii) all of the $B_{i-1,j}^+$'s (resp., $B_{i,j}^-$'s) have the same number of 0's to within 2.

If either Case 1 or Case 2 holds after stage (c), note that the following claims hold after stage (e), $0 < i < m$: (i) every C_i^j is sorted, and (ii) all of the C_i^j's have the same number of 0's to within 4.

We now define the d-permutation π' of stage (f) so that: (i) the wires in B_0^- and B_{m-1}^+ are left alone, and (ii) for each i, $0 \le i < m-1$, the C_i^j's are interleaved (in place) by applying the $(b+1)$-permutation $\hookleftarrow_{b+1}^{b'+1}$.

Let $C_0 = B_0^-$, $C_m = B_{m-1}^+$, and assume that either Case 1 or Case 2 held after stage (c). Then the following conditions hold after stage (f): (i) C_i has a dirty region of size at most $4 \cdot \mathrm{pow}(b'+1) = \mathrm{pow}(b'+3)$, $0 \le i \le m$, and (ii) no two non-consecutive C_i's are non-trivial. If 0 or 1 of the C_i's are non-trivial then A is $(b'+3)$-sorted, and we are done. Otherwise, we can assume that C_i and C_{i+1} are non-trivial for some particular i, $0 \le i < m$. It follows easily that Case 2 held after stage (c), and that the output of A after stage (f) is the same as after stage (c); hence, the dirty region of A has size at most $\mathrm{pow}(b'+1)$ after stage (f). □

Lemmas 4.4, 5.17, and 6.3 together imply that

$$\mathit{Sort}_D(d, a, \lfloor \gamma \cdot a \rfloor + 1, O(\mathrm{pow}(a) \cdot \varepsilon)) \le 2 \cdot a - \lfloor \gamma \cdot a \rfloor$$

for any admissible triple (γ, ε, a) and all d such that $0 \le a \le d$. Lemma 5.6 implies that for any function $\varepsilon(a) = \mathrm{pow}(-\mathrm{pow}(o(a)))$, there is a function $f(a) = o(1)$ such that $(\gamma_c + f(a), \varepsilon, a)$ is an admissible triple for all $a \ge 0$. Hence,

$$\mathit{Sort}_D(d, a, \lfloor \gamma(a) \cdot a \rfloor + 1, \mathrm{pow}(-\mathrm{pow}(o(a)))) \le 2 \cdot a - \lfloor \gamma(a) \cdot a \rfloor, \tag{11}$$

with $\gamma(a) = \gamma_c + o(1)$. Substituting the bounds of Equation (11) (with $a = b+2$) and Lemma 4.3 (with $(a, d) = (b - \lfloor \gamma \cdot (b+2) \rfloor + 1, d)$) into the inequality of Lemma 6.4, we find that

$$\begin{aligned} & \mathit{Sort}_D(d, a, \lfloor (\gamma_c + o(1)) \cdot (b+2) \rfloor + 3, \mathrm{pow}(-\mathrm{pow}(o(b))) + \varepsilon') \\ \le \quad & \mathit{Sort}_D(d, a, b, \varepsilon') + (3 - 2 \cdot \gamma_c + o(1)) \cdot b, \end{aligned}$$

for all ε' in $[0, 1]$. By starting with Equation (11), and then iteratively applying the preceding inequality (with $b \approx \gamma \cdot a, \gamma^2 \cdot a, \gamma^3 \cdot a, \ldots$) until Equation (10) can be "inexpensively" applied (e.g., with $b = o(a)$), we find that

$$\mathit{Sort}_D(d, a, \mathrm{pow}(-\mathrm{pow}(o(a)))) \le \frac{2 - \gamma_c^2 + o(1)}{1 - \gamma_c} \cdot a,$$

for all a and d such that $0 \le a \le d$. Using Lemma 4.8 to eliminate the random aspects of the preceding construction, the proof of Lemma 6.1 is now complete. (We remark that randomization has not been used in the operation phase of any level in our construction. Furthermore, the only non-trivial probability distributions used in the permutation phase of any level are the $\Pi_R(d, a)$ distributions, $0 < a \le d$.)

Note that we have used the AKS sorting network as part of our construction. It should be emphasized, however, that the AKS sorting network is only used to allow the function $\varepsilon(d)$ of Lemma 6.1 to be set as small as possible. For example, one could prove Lemma 6.1 with $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(o(\sqrt{d})))$ by cutting off the preceding recurrence at $b = o(\sqrt{d})$ and applying bitonic sort, instead of cutting it off at $b = o(d)$ and applying the AKS sorting network.

7. An Optimal-Depth Hypercubic Network that Sorts Most Inputs. In this section, we establish the existence of a depth-$O(d)$, hypercubic d-network that sorts most inputs. We once again make use of the high-level strategy described at the beginning of §6, except that we make use of Lemma 5.17 instead of Lemma 5.16.

In contrast with §6, however, we do not concern ourselves with constant factor issues. This leads to a much simpler construction. In particular, we do not require a hypercubic analogue of Lemma 6.4.

LEMMA 7.1. *Let $\psi(a)$ be any function such that $Sort_D^h(d, \psi(a), 0) = O(d)$, $0 \le a \le d$. For each function $\varepsilon(d) = \text{pow}(-\text{pow}(\psi(d)))$, we have*

$$Sort_D^h(d, a, \varepsilon(a)) = O(a).$$

Furthermore, there is a deterministic d-network that achieves this bound. □

By Lemma 4.5, the function ψ appearing in the statement of Lemma 7.1 is $\Omega(\sqrt{d})$. In fact, by Theorem 9.1, we have

$$\psi(d) = \Omega\left(\frac{d}{\text{pow}(\sqrt{\upsilon_c \cdot \lg d}) \cdot \lg d}\right). \tag{12}$$

The following theorem provides an interpretation of Lemma 7.1 in the permutation domain.

THEOREM 7.1. *Let the function ψ be as defined in Lemma 7.1. For each function $\varepsilon(d) = \text{pow}(-\text{pow}(\psi(d)))$, there is a deterministic hypercubic d-network of depth $O(d)$ that sorts a random d-permutation drawn from $\Pi_R(d)$ with probability at least $1 - \varepsilon(d)$.*

Proof. Similar to the proof of Theorem 6.1. □

LEMMA 7.2. *For all a, b, and d such that $0 \le b \le a \le d$, and all ε and ε' in $[0, 1]$, we have*

$$Sort_D^h(d, a, \varepsilon + 2 \cdot \varepsilon') \le Sort_D^h(d, a, b, \varepsilon) + Sort_D^h(d, b, \varepsilon') + 2 \cdot Merge_D^h(d, b, 1) + O(a).$$

Proof. Similar to the proof of Lemma 6.2. The only difference is that we use an $O(a)$-depth hypercubic (d, a)-network (guaranteed to exist by Lemma 4.7) to implement each of the d-permutations of stages (b), (e), (g), and (h). This accounts for the additive $O(a)$ term on the right-hand side of the inequality. □

LEMMA 7.3. *For all a, b, and d such that $0 \le b \le a \le d$, and all ε in $[0, 1]$, we have*

$$Sort_D^h(d, a, b + 1, \varepsilon) \le Most_D^h(d, a, b, \varepsilon) + Insert_D^h(d, a - b) + O(a).$$

Proof. Similar to the proof of Lemma 6.3. The only difference is that we use an $O(a)$-depth hypercubic (d, a)-network (guaranteed to exist by Lemma 4.7) to implement each of the d-permutations of stages (b), (c), and (e). This accounts for the additive $O(a)$ term on the right-hand side of the inequality. □

Lemmas 4.4, 5.17, and 7.3 together imply that

$$Sort_D^h(d, a, \lfloor \gamma \cdot a \rfloor + 1, O(\text{pow}(a) \cdot \varepsilon)) = O(a) \tag{13}$$

for all d and any admissible triple (γ, ε, a) such that $0 \le a \le d$. Let δ be any constant, $0 < \delta < 1 - \gamma_c$. By Lemma 5.6, there is a function $g(d) = \Theta(d)$ such that $(\gamma_c + \delta, \text{pow}(-\text{pow}(g(a))), a)$ is an admissible triple for all $a \ge 0$. Hence,

$$Sort_D^h(d, a, \lfloor \gamma \cdot a \rfloor + 1, \text{pow}(-\text{pow}(\Theta(a)))) = O(a),$$

with $\gamma = \gamma_c + \delta$, and $0 \le a \le d$. Lemmas 4.3 and 7.2 (with $b = \lfloor \gamma \cdot a \rfloor + 1$) now give

$$Sort_D^h(d, a, \mathrm{pow}(-\mathrm{pow}(\Theta(a))) + 2 \cdot \varepsilon') \le Sort_D^h(d, \lfloor \gamma \cdot a \rfloor + 1, \varepsilon') + O(a)$$

for all ε' in $[0, 1]$. Iteratively applying the preceding inequality, we find that

$$Sort_D^h(d, a, \mathrm{pow}(-\mathrm{pow}(\Theta(b)))) \le Sort_D^h(d, b, 0) + O(a)$$

for all a, b, and d such that $0 \le b \le a \le d$. Substituting $\psi(a)$ for b, where the function ψ is as defined in the statement of Lemma 7.1, we obtain

$$Sort_D^h(d, a, \mathrm{pow}(-\mathrm{pow}(\psi(a)))) = O(a),$$

for all a and d such that $0 \le a \le d$. Using Lemma 4.8 to eliminate the random aspects of the preceding construction, the proof of Lemma 7.1 is now complete. (We remark that randomization has not been used in the operation phase of any level in our construction. Furthermore, the only use of randomization in the permutation phase arises from applying Lemma 4.7 to implement random d-permutations drawn from $\Pi_R(d, a)$, $0 < a \le d$.)

8. Deterministic Merging. Many sorting algorithms, both sequential as well as parallel, are based on merging. For instance, sequential merge sort and Batcher's bitonic sorting network are both based on 2-way merging. Since merging two sorted lists of length $\mathrm{pow}(d)$ requires $\Omega(d)$ depth, one cannot hope to obtain a $o(d^2)$-depth sorting network (hypercubic or otherwise) by repeated 2-way merging. This section describes how to use a network $\mathcal{N}$ that sorts most inputs to construct a high-order merging network $\mathcal{N}'$, that is, an m-way merging network for some $m \gg 2$. A similar technique has recently been used by Ajtai, Komlós, and Szemerédi as part of an improved version of their original sorting network construction [3]. The multiplicative constant associated with the new construction is significantly lower than the constant established by Paterson [15], though it remains impractical.

The main idea underlying the results of this section may be informally outlined as follows. An n-input network is an m-way merging network if and only if it correctly merges all possible input vectors consisting of m sorted vectors of length n/m. But as in the case of sorting networks, we can easily prove a 0-1 principle for merging networks: Any n-input network that correctly merges every input vector consisting of m sorted 0-1 vectors of length n/m is an m-way merging network. A key observation is that the total number of 0-1 vectors which can be obtained by concatenating m sorted 0-1 vectors of length n/m is $(n/m+1)^m$, which is fairly small (compared to 2^n, the total number of 0-1 vectors, for example) when m is not too large. As a result, we can use an averaging argument to prove that, given any network $\mathcal{N}$ which for all k sorts a sufficiently high fraction (dependent on m) of the $\binom{n}{k}$ 0-1 vectors with k 0's and $n-k$ 1's, there exists a permutation π such that the network $\mathcal{N}'$ obtained by composing π with $\mathcal{N}$ sorts *all* of the $(n/m+1)^m$ 0-1 vectors consisting of m sorted 0-1 vectors of length n/m; in other words, $\mathcal{N}'$ is an m-way merging network with the same size and depth as network $\mathcal{N}$.

LEMMA 8.1. *Let $\mathcal{N}$ denote a (hypercubic) (d, a)-network that sorts each a-cube with probability at least $1 - \varepsilon$ on any a-random 0-1 input d-vector, $m = \mathrm{pow}(d - a)$, Φ denote a subset of $\Phi(d)$, $\Phi^{(i)} \subseteq \Phi(a)$ denote the projection of the ith a-cube of Φ onto $\Phi(a)$, $0 \le i < m$, and $\Phi' = \cup_{0 \le i < m} \Phi^{(i)}$. If $|\Phi'| < 1/\varepsilon$, then there exists a d-permutation π in $\Pi(d, a)$ such that the (hypercubic) (d, a)-network $\mathcal{N}'$ obtained by* *composing π with $\mathcal{N}$ satisfies*

$$\Phi \subseteq Sort(\mathcal{N}', a).$$

Proof. By definition, (hypercubic) (d, a)-network $\mathcal{N}$ can be partitioned along input a-cubes into m disjoint, identical a-networks $\mathcal{N}_a$. Thus, it is sufficient to construct an a-permutation π' such that the (hypercubic) a-network $\mathcal{N}_a'$ obtained by composing π' with $\mathcal{N}_a$ satisfies

$$\Phi' \subseteq Sort(\mathcal{N}_a').$$

To determine a suitable a-permutation π', we construct the undirected bipartite graph with vertex sets $U = \Phi'$ and $V = \Pi(a)$, and an edge from vertex ϕ in U to vertex π in V if and only if the 0-1 a-vector ϕ is sorted by the a-network obtained by composing a-permutation π with $\mathcal{N}_a$. The degree of every vertex in U is at least $(1-\varepsilon) \cdot (\text{pow}(a))!$, and so the sum of the degrees of the vertices in U is at least $(1-\varepsilon) \cdot |U| \cdot (\text{pow}(a))!$. This sum is identical to that attained over V, so some vertex π' in V has degree at least $(1-\varepsilon) \cdot |U| > |U| - 1$. Since the degree of π' is an integer, vertex π' is connected to every vertex in U. Thus, the a-network obtained by composing a-permutation π' with network $\mathcal{N}_a$ sorts every element of Φ'. ∎

LEMMA 8.2. *For nonnegative integers a' and b', let $\mathcal{N}$ be defined as in Lemma 8.1 with $a = a' + b'$ and*

$$(\text{pow}(a') + 1)^{\text{pow}(b')} < 1/\varepsilon.$$

Let Φ denote the set of all 0-1 d-vectors ϕ such that each a-cube of ϕ belongs to $\Phi_M(a', b')$. Then there exists a d-permutation π in $\Pi(d, a)$ such that the (hypercubic) (d, a)-network $\mathcal{N}'$ obtained by composing π with $\mathcal{N}$ satisfies

$$\Phi \subseteq Sort(\mathcal{N}', a).$$

Proof. We can apply Lemma 8.1 (with $\Phi' = \Phi_M(a', b')$) since

$$\begin{aligned} |\Phi_M(a', b')| &= (\text{pow}(a') + 1)^{\text{pow}(b')} \\ &< 1/\varepsilon. \end{aligned}$$

∎

Note that in Lemmas 8.1 and 8.2, the depth of $\mathcal{N}'$ only exceeds that of $\mathcal{N}$ by the depth required to implement a d-permutation in $\Pi(d, a)$. By Lemma 4.7, this additional depth is at most $2 \cdot a$ for a hypercubic construction. (For a non-hypercubic construction, no additional depth is required to implement a fixed permutation.)

9. A Near-Optimal Hypercubic Sorting Network. In this section, we construct a hypercubic sorting network with nearly logarithmic depth. At a high level, the construction is simply based on recursive high-order merging: The input is partitioned into some number of equal-sized lists, each of these lists is sorted recursively, and the resulting set of sorted lists are merged together. The recursion is cut off by applying bitonic sort on subproblems that are sufficiently small. The primary question that remains to be addressed is how to perform the merge step efficiently.

Lemmas 9.1 and 9.2 make use of the results of §8 (specifically, Lemma 8.2) to reduce the merge step to a smaller sorting problem.

LEMMA 9.1. *Let $\nu(d)$ be any function such that $\nu(d) = \omega(1)$ and $\nu(d) = o(d)$, and let $\varepsilon(d) = \text{pow}(-\text{pow}(\mu(d) \cdot d))$ where $\mu(d) = \mu_c - \frac{1}{\nu(d)}$. For all a, b, and d such that $0 \leq b \leq a \leq d$, we have*

$$Sort_D^h(d, a, O(\text{pow}(b) \cdot \varepsilon(b))) \leq Sort_D^h(d, b, 0) + O(a \cdot \nu(a)).$$

Proof. By Lemma 5.7, there exists a function $\gamma(d) = 1 - \frac{3-o(1)}{4 \cdot \nu(d)}$ such that $(\gamma(a), \varepsilon(a), a)$ is an admissible triple for all $a \geq 0$. For such an admissible triple, Equation (13) then implies

$$Sort_D^h(d, a, \lfloor \gamma(a) \cdot a \rfloor + 1, O(\text{pow}(a) \cdot \varepsilon(a))) = O(a).$$

Lemmas 4.3 and 7.2 now give

$$Sort_D^h(d, a, O(\text{pow}(a) \cdot \varepsilon(a)) + 2 \cdot \varepsilon') \leq Sort_D^h(d, \lfloor \gamma(a) \cdot a \rfloor + 1, \varepsilon') + O(a),$$

for all ε' in $[0, 1]$. Iteratively applying the preceding inequality, we find that

$$Sort_D^h(d, a, O(\text{pow}(b) \cdot \varepsilon(b))) \leq Sort_D^h(d, b, 0) + O(a \cdot \nu(a))$$

for all a, b, and d such that $0 \leq b \leq a \leq d$. $\square$

LEMMA 9.2. *Let the function ν be as defined in Lemma 9.1, and let $\mu(d) = \mu_c - \frac{2}{\nu(d)}$. Then*

$$Merge_D^h(d, a - \lfloor \mu(b) \cdot b \rfloor, \lfloor \mu(b) \cdot b \rfloor) \leq Sort_D^h(d, b, 0) + O(a \cdot \nu(a))$$

for all a, b, and d such that $(2 + \lg a) \cdot \nu(b) \leq b \leq a \leq d$.

Proof. Let $\varepsilon(d) = \text{pow}(-\text{pow}(\mu'(d) \cdot d))$ where $\mu'(d) = \mu_c - \frac{1}{\nu(d)}$. By Lemma 9.1,

$$Sort_D^h(d, a, O(\text{pow}(b) \cdot \varepsilon(b))) \leq Sort_D^h(d, b, 0) + O(a \cdot \nu(a))$$

for all a, b, and d such that $0 \leq b \leq a \leq d$. Hence, for b sufficiently large, there exists a hypercubic (d, a)-network $\mathcal{N}$ of depth $Sort_D^h(d, b, 0) + O(a \cdot \nu(a))$ that sorts each a-cube with probability at least $1 - \varepsilon'$ on any a-random 0-1 input d-vector, where

$$\begin{aligned} \varepsilon' &= \text{pow}(2 \cdot b) \cdot \varepsilon(b) \\ &= \text{pow}(\text{pow}(1 + \lg b) - \text{pow}(\mu'(b) \cdot b)) \\ &< \text{pow}(-\text{pow}(\mu'(b) \cdot b - 1)). \end{aligned}$$

The result now follows from Lemma 8.2 since

$$\begin{aligned} (\text{pow}(a - \lfloor \mu(b) \cdot b \rfloor) + 1)^{\text{pow}(\lfloor \mu(b) \cdot b \rfloor)} &< \text{pow}(2 \cdot a \cdot \text{pow}(\mu(b) \cdot b)) \\ &= \text{pow}(\text{pow}(\mu(b) \cdot b + 1 + \lg a)) \\ &= \text{pow}(\text{pow}(\mu'(b) \cdot b + 1 + \lg a - b/\nu(b))) \\ &\leq \text{pow}(\text{pow}(\mu'(b) \cdot b - 1)) \\ &< 1/\varepsilon'. \end{aligned}$$

$\square$

As a consequence of the preceding lemma, we can develop a recurrence for $Sort_D^h(d,a,0)$. Note that

$$Sort_D^h(d,a,0) \le \min_{0\le b\le a} Sort_D^h(d,a-b,0) + Merge_D^h(d,a-b,b)$$

for all a, b, and d such that $0 \le b \le a \le d$. (This inequality is immediate, since we can always sort a-cubes by: (i) sorting $(a-b)$-cubes, and (ii) merging the sorted $(a-b)$-cubes within each a-cube.) Let the functions ν and μ be as defined in Lemma 9.2. Applying Lemma 9.2 to the preceding inequality, we obtain

$$Sort_D^h(d,a,0) \le \min_{(2+\lg a)\cdot\nu(b)\le b\le a} Sort_D^h(d,a-\lfloor\mu(b)\cdot b\rfloor,0)+Sort_D^h(d,b,0)+O(a\cdot\nu(a))$$

for all a, b, and d such that $0 \le b \le a \le d$. Fixing d and letting $S(a) = Sort_D^h(d,a,0)$, we can write this recurrence more simply as

$$S(a) \le \min_{(2+\lg a)\cdot\nu(b)\le b\le a} S(a-\lfloor\mu(b)\cdot b\rfloor) + S(b) + O(a\cdot\nu(a)).$$

In Appendix A it is proven that

$$S(a) = O(a\cdot \mathrm{pow}(\sqrt{\upsilon_c\cdot\lg a})\cdot\lg a).$$

(The constant υ_c is defined in Equation (9).) The preceding bound is proven with $\nu(d) = \Theta(\sqrt{\lg d})$, and seems to be the best upper bound obtainable using this recurrence. Setting $a = d$, we obtain a proof of the following theorem.

THEOREM 9.1. *For all $d \ge 0$, we have*

$$Sort_D^h(d,0) = O(d\cdot \mathrm{pow}(\sqrt{\upsilon_c\cdot\lg d})\cdot\lg d).$$

□

10. An Optimal Randomized Hypercubic Sorting Algorithm. In §7, we constructed a depth-$O(d)$ hypercubic sorting network that sorts most d-permutations. In the present section, we modify that result to obtain a polynomial-time uniform $O(d)$-depth coin-tossing hypercubic network that sorts *every* d-permutation (and hence, every d-vector) with high probability. We then use this coin-tossing network to develop a polynomial-time uniform hypercubic algorithm that sorts every d-vector in $O(d)$ time with high probability.

We define a *hypercubic algorithm* as any normal hypercube algorithm. (See [12, §3.1.3], for example, for a definition of the class of normal hypercube algorithms.) Every depth-a hypercubic sorting d-network corresponds to a (possibly non-uniform) hypercubic sorting algorithm that runs in $O(a)$ time on any pow(d)-processor hypercubic machine. Of course, the converse is not true in general; most of the basic operations allowed within a normal hypercube algorithm (e.g., the usual set of arithmetic operations) cannot be performed by a hypercubic sorting network.

A sorting network is hard-wired, and has a fixed depth or "running time", that is independent of the input. On the other hand, a sorting algorithm can have an arbitrarily large gap between its worst-case and average-case running times. For example, consider a sorting algorithm with the following structure:

1. Apply a random d-permutation drawn from $\Pi_R(d)$ to the input d-vector.
2. Attempt to sort the resulting d-vector using a time-$T(d)$ method that correctly sorts most d-permutations.

3. Check whether Step 2 was successful. If so, halt. If not, return to Step 1.

The worst-case running time of such an algorithm is infinite, while the average-case running time could be as low as $O(T(d))$. One might attempt to develop a hypercubic sorting algorithm with this structure by using the d-network of Theorem 7.1 to implement Step 2 with $T(d) = O(d)$. Step 3 is trivial to implement in $O(d)$ time. However, two difficulties remain to be addressed.

The first difficulty is that Step 1 is not easily implemented by a hypercubic algorithm. We will overcome this difficulty by making use of a depth-d shuffle-"?" d-network to randomly permute the input data. Although the d-permutation π applied by such a d-network is not d-random, we prove in Lemma 10.2 below that π is sufficiently random for our purposes.

The second difficulty is that the hypercubic algorithm corresponding to Theorem 7.1 is not polynomial-time uniform. This undesirable characteristic stems from our use of Lemma 4.8 to remove the random aspects of our hypercubic network construction. As discussed at the end of §7, there is only one source of randomness in our construction: Whenever we apply the shuffle-"+" (d, a)-network of Lemma 5.17, we first apply an a-random d-permutation. We overcome the second difficulty by replacing such a-random d-permutations with a depth-a unshuffle-"0" d-network followed by a depth-a shuffle-"?" d-network.

Lemmas 10.2 and 10.3 below prove that we can approximately sort *every* d-permutation with high probability by applying a depth-d shuffle-"?" d-network followed by a depth-$\lceil d/2 \rceil$ shuffle-"+" d-network. Lemma 10.1 widens the range of input distributions for which the analysis of §5 can be applied. (Recall that the σ_α functions were defined in §5.1.)

LEMMA 10.1. *Let $\mathcal{N}$ denote a coin-tossing d-network, and assume that for each length-d binary string α, input wire α is set to 0 with probability p_α, and to 1 otherwise. Further assume that the set of events $E_\alpha \stackrel{\text{def}}{=}$ "input wire α receives a 0" are mutually independent. Then the output wire with index α receives a 0 with probability at least $\sigma_\alpha(p^-)$ and at most $\sigma_\alpha(p^+)$, where $p^- = \min_\alpha p_\alpha$ and $p^+ = \max_\alpha p_\alpha$.*

Proof. Immediate from Lemma 4.11. □

LEMMA 10.2. *Let a, b, and d denote nonnegative integers such that $d = a + b$, (γ, ε, a) be an admissible triple,*

$$\begin{aligned} \delta &= (1 - 2 \cdot \varepsilon) \cdot \mathrm{pow}((\gamma - 1) \cdot a)/4, \\ \varepsilon' &= 2 \cdot e^{-2 \cdot \delta^2 \cdot \mathrm{pow}(b)}, \end{aligned}$$

and $\mathcal{N}$ denote the depth-$(a+d)$ d-network obtained by composing: (i) a depth-d shuffle-"?" d-network, and (ii) a depth-a shuffle-"+" d-network. Then there exists a set A of at least $\mathrm{pow}(d) - \mathrm{pow}(\gamma \cdot a + b)$ output wires of $\mathcal{N}$, and a fixed permutation π of A, such that the following condition holds with probability at least $1 - \mathrm{pow}(d) \cdot \varepsilon - \mathrm{pow}(a) \cdot \varepsilon'$ after execution of $\mathcal{N}$ on any 0-1 input d-vector ϕ: If permutation π is applied to A, then the resulting length-$|A|$ 0-1 output vector is $(\gamma \cdot a + b)$-sorted.

Proof. Let 0-1 vector ϕ in $\Phi(d, k)$ be input to d-network $\mathcal{N}$, and set $p = k/\mathrm{pow}(d)$. Throughout this proof, the symbols α and β will be used to denote binary strings of length a and b, respectively. A *random execution* will refer to an execution of d-network $\mathcal{N}$ on input ϕ.

For each β, define $C_0(\beta)$ (resp., $C_2(\beta)$, $C_3(\beta)$) as the set of $\mathrm{pow}(a)$ level-0 input wires (resp., level-d input wires, level-$(a + d - 1)$ output wires) with indices of the form $\alpha\beta$ (resp., $\alpha\beta$, $\beta\alpha$) for some α. For each α, define $C_1(\alpha)$ as the set of $\mathrm{pow}(b)$ level-$(a - 1)$ output wires with indices of the form $\beta\alpha$ for some β.

For each β, define p_β to be the fraction of 0's induced by input ϕ on $C_0(\beta)$ (i.e., the number of 0's assigned to $C_0(\beta)$ divided by $\operatorname{pow}(a)$). Note that $p = (\sum_\beta p_\beta)/\operatorname{pow}(b)$. For each α, let X_α denote the random variable corresponding to the number of 0's received by the wires of $C_1(\alpha)$ in a random execution, and let $q_\alpha = X_\alpha/\operatorname{pow}(b)$. Note that, unlike the p_β's, each q_α is a random variable. Furthermore, the random variable X_α is easily seen to be the sum of $\operatorname{pow}(b)$ independent Bernoulli trials, where trial β has success probability p_β. Thus, a standard Chernoff-type argument [6] implies

$$\Pr\{|X_\alpha - p \cdot \operatorname{pow}(b)| \geq \vartheta \cdot \operatorname{pow}(b)\} \leq 2 \cdot e^{-2\cdot\vartheta^2 \cdot \operatorname{pow}(b)} \tag{14}$$

for all $\vartheta \geq 0$. Define a random execution to be *δ-balanced* if

$$p - \delta \leq q_\alpha \leq p + \delta$$

for all α. By Equation (14), a random execution is δ-balanced with probability at least $1 - \operatorname{pow}(a) \cdot \varepsilon'$ (set $\vartheta = \delta$.)

Note that the last a levels of d-network $\mathcal{N}$ form a (d, a)-network. Hence, these levels can be partitioned into $\operatorname{pow}(b)$ disjoint depth-a shuffle-"+" a-networks $\mathcal{N}_\beta$, where the input and output wires of $\mathcal{N}_\beta$ correspond to $C_2(\beta)$ and $C_3(\beta)$, respectively. Let $E_{\alpha\beta}$ denote the event that input α of $\mathcal{N}_\beta$ (i.e., level-d input wire $\alpha\beta$ of $\mathcal{N}$) receives a 0 in a random execution. Let $f_{\alpha\beta}(p)$ denote the probability that $E_{\alpha\beta}$ occurs in a random δ-balanced execution. Let $g_{\beta\alpha}(p)$ denote the probability that output α of $\mathcal{N}_\beta$ (i.e., output $\beta\alpha$ of $\mathcal{N}$) receives a 0 in a random δ-balanced execution. Note that $f_{\alpha\beta}(p) = q_\alpha$, since wire α of $C_2(\beta)$ receives the value of a wire chosen uniformly at random from $C_1(\alpha)$. Furthermore, since the sets $C_1(\alpha)$ are mutually disjoint, we find that for each β and for each fixed setting of the q_α values, the $\operatorname{pow}(a)$ events $E_{\alpha\beta}$ are mutually independent. Lemma 10.1 can therefore be applied to each a-network $\mathcal{N}_\beta$, and yields

$$\sigma_\alpha(p - \delta) \leq g_{\beta\alpha}(p) \leq \sigma_\alpha(p + \delta),$$

for all α and β.

Define A to be the set (guaranteed to exist by Lemma 5.13) of at least $\operatorname{pow}(d) - \operatorname{pow}(\gamma \cdot a + b)$ output wires of $\mathcal{N}$ indexed by length-d binary strings $\beta\alpha$ such that

$$\Gamma_\alpha(1 - \varepsilon) - \Gamma_\alpha(\varepsilon) \leq \delta.$$

We set π to the permutation of set A that sorts the $\Gamma_\alpha(\varepsilon)$ values in ascending order. Ties may be broken arbitrarily. (As discussed in the proof of Lemma 5.14, the set A and permutation π can be computed efficiently.) It remains to prove that our choice of A and π satisfies the requirements of the lemma.

Let $p^- = \max\{0, p - 2 \cdot \delta\}$, $p^+ = \min\{1, p + 2 \cdot \delta\}$, and B denote the set of binary strings α in A for which $\Gamma_\alpha(\varepsilon)$ is contained in $[p^-, p + \delta]$. Because the σ_α's are monotonically increasing, and using linearity of expectation, we have

$$\begin{aligned}
\sum_\alpha |\sigma_\alpha(p^+) - \sigma_\alpha(p^-)| &= \sum_\alpha (\sigma_\alpha(p^+) - \sigma_\alpha(p^-)) \\
&= \left(\sum_\alpha \sigma_\alpha(p^+)\right) - \left(\sum_\alpha \sigma_\alpha(p^-)\right) \\
&= (p^+ - p^-) \cdot \operatorname{pow}(a) \\
&\leq 4 \cdot \delta \cdot \operatorname{pow}(a).
\end{aligned}$$

For each α in B we have $\sigma_\alpha(p^-) \le \varepsilon$, $\sigma_\alpha(p^+) \ge 1-\varepsilon$, and hence

$$|\sigma_\alpha(p^+) - \sigma_\alpha(p^-)| \ge 1 - 2\cdot\varepsilon.$$

The preceding inequalities imply that

$$\begin{aligned} |B| &\le 4\cdot\delta\cdot \mathrm{pow}(a+b)/(1-2\cdot\varepsilon) \\ &= \mathrm{pow}(\gamma\cdot a+b). \end{aligned}$$

Note that the set of binary strings B is mapped to a contiguous interval by permutation π. Let A^- (resp., A^+) denote the set of all binary strings in $A\setminus B$ mapped to positions lower (resp., higher) than B by π.

Let $\beta\alpha$ denote the binary string associated with an arbitrary output in A^-. Thus $\Gamma_\alpha(\varepsilon) < p^-$ (so $p^- > 0$ and $p-\delta > 0$), which implies $\Gamma_\alpha(1-\varepsilon) < p-\delta$ and hence $\sigma_\alpha(p-\delta) > 1-\varepsilon$. Combining this inequality with the lower bound of Equation (10), we find that $g_{\beta\alpha}(p) > 1-\varepsilon$. (The probability that output $\beta\alpha$ receives a 1 is less than ε.)

Similarly, let α denote the binary string associated with some output in A^+. Thus $\Gamma_\alpha(\varepsilon) > p+\delta$ (so $p+\delta < 1$), which implies $\sigma_\alpha(p+\delta) < \varepsilon$. Combining this inequality with the upper bound of Equation (10), we find that $g_{\beta\alpha}(p) < \varepsilon$. (The probability that output $\beta\alpha$ receives a 0 is less than ε.)

We conclude that if permutation π is applied to A, the resulting length-$|A|$ 0-1 vector has a dirty region of size at most $|B| = \mathrm{pow}(\gamma\cdot a+b)$ with probability at least $1-\mathrm{pow}(d)\cdot\varepsilon - \mathrm{pow}(a)\cdot\varepsilon'$. □

LEMMA 10.3. *Let $\mathcal{N}$, A, and π be defined as in Lemma 10.2, with $a = \lceil d/2\rceil$ and $b = \lfloor d/2\rfloor$. Then there exist constants γ and ε in $(0,1)$ such that the following condition holds with probability at least $1 - O(\mathrm{pow}(-\mathrm{pow}(\varepsilon\cdot d)))$ after execution of $\mathcal{N}$ on any 0-1 input vector ϕ: If permutation π is applied to A, then the resulting length-$|A|$ 0-1 output vector is $(\gamma\cdot d)$-sorted.*

Proof. This is a straightforward consequence of Lemmas 5.6 and 10.2. □

Given Lemma 10.3, we can easily prove an analogue of Lemma 7.3 that uses "?" gates instead of random permutations. (It is important to note that the (d,a)-network associated with the construction of Lemma 10.3 is composed of the following four stages: (a) a depth-a unshuffle-"0" d-network, (b) a depth-a shuffle-"?" d-network, (c) a depth-$\lceil a/2\rceil$ unshuffle-"0" d-network, and (d) a depth-$\lceil a/2\rceil$ shuffle-"+" d-network.) We can then use the scheme of §7 to prove Theorem 10.1 below with

$$\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(\Theta(\sqrt{d}))).$$

Unfortunately, we cannot take advantage of the improvement associated with Equation 12 because the construction of §9 is not polynomial-time uniform.

THEOREM 10.1. *For each function $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(O(\sqrt{d})))$, there is a polynomial-time uniform $O(d)$-depth coin-tossing hypercubic d-network that sorts any input d-vector probability at least $1-\varepsilon(d)$.* □

The scheme of §7 can also be used to prove Theorem 10.1 below with the function ε as defined in Theorem 10.2. In this case, we can dramatically decrease the failure probability by making use of the Sharesort algorithm of Cypher and Plaxton [9]. Sharesort is a polynomial-time uniform hypercubic sorting algorithm with worst-case running time $O(d\cdot \lg^2 d)$ [9]. Note that Sharesort runs in $O(d)$ time on $O(d/\lg^2 d)$-cubes. Hence, we can modify the scheme of §7 by cutting off the sorting recurrence at $\Theta(d/\lg^2 d)$-cubes instead of $\Theta(\sqrt{d})$-cubes (as allowed by bitonic sort). Unfortunately, Sharesort does not correspond to a hypercubic sorting network since, for example: (i) Sharesort makes copies of keys, and (ii) Sharesort performs a variety of arithmetic operations on auxiliary integer variables. For these reasons, we have not been able to make use of Sharesort in previous sections of the paper.

A small improvement to the Sharesort bound is known when polynomial-time "pre-processing" (to compute certain look-up tables) is allowed. In particular, the running time of Sharesort can be improved to $O(d \cdot (\lg d) \cdot \lg^* d)$ in that case [8]. This improvement has been incorporated into the $\varepsilon(d)$ bound of Theorem 10.2. If exponential pre-processing is allowed, the running time of Sharesort can be improved further to $O(d \cdot \lg d)$ [8]. However, it is not clear whether the latter result could be used to improve the $\varepsilon(d)$ bound of Theorem 10.2. (The lack of uniformity can be eliminated through randomization. However, the failure probability of the resulting algorithm seems to be strictly higher than that given by Theorem 10.2.)

THEOREM 10.2. *Let* $f(d) = \Theta\left(\frac{d}{(\lg d) \cdot \lg^* d}\right)$. *For each function* $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(f(d)))$, *there is a polynomial-time uniform randomized hypercubic sorting algorithm that runs in* $O(d)$ *time (on any input* d*-vector) with probability at least* $1 - \varepsilon$. □

11. An Optimal Bit-Serial Randomized Hypercubic Sorting Algorithm. An *order-*d *omega machine* is a $((d+1) \cdot \mathrm{pow}(d))$-processor machine, $d \geq 0$. Each processor has an associated ID of the form (i, j), $0 \leq i \leq d$, $0 \leq j < \mathrm{pow}(d)$. We define the *ith level* of a given order-d omega machine as the set of $\mathrm{pow}(d)$ processors with IDs of the form (i, j), $0 \leq j < \mathrm{pow}(d)$. The processors of an order-d omega machine are interconnected according to the following rules:

1. There is no wire between any pair of processors in non-consecutive levels.
2. For all i such that $0 \leq i < d$, there is a wire connecting processor (i, j) to processor $(i+1, j')$ if and only if $j_k = j'_{k+1}$, $0 \leq k < d-1$.

Omega machines belong to the class of butterfly-like machines discussed in [12, §3.8.1].

Observe that there is a close correspondence between an order-d omega machine $\mathcal{M}$ and a depth-d shuffle d-network $\mathcal{N}$. In particular, consider the 1-1 function $f(i, j)$ that maps processor (i, j) of $\mathcal{M}$ to: (i) level-i input wire j of $\mathcal{N}$, $0 \leq i < d$, and (ii) level-$(i-1)$ output wire j of $\mathcal{N}$, $0 < i \leq d$. (Recall that level-i input wire j and level-$(i-1)$ output wire j represent the same wire, $0 < i < d$. Hence, f is indeed a function.) Then there is a wire between processors (i, j) and $(i+1, j')$ in $\mathcal{M}$ if and only if wires $f(i, j)$ and $f(i+1, j')$ in $\mathcal{N}$ are connected to a common gate x, with $f(i, j)$ as an input wire and $f(i+1, j')$ as an output wire.

In the bit model, it is assumed that a processor can only perform one bit operation per time step. Thus, b bit steps are required to send a b-bit message to an adjacent processor. Similarly, b bit steps are required to compare two b-bit operands located at the same processor. In this section, we provide a bit-serial polynomial-time uniform randomized algorithm for sorting $\mathrm{pow}(d)$ $O(b)$-bit records on an order-d omega machine in $O(b + d)$ bit steps. This time bound is easily seen to be optimal. For $b = \Omega(d)$, the processor bound is also optimal. Our algorithm can be adapted to achieve the same asymptotic performance on any butterfly-like machine.

DEFINITION 11.1. *A* bit-serial omega emulation scheme *for a depth-*a *word-size-*b d*-network* $\mathcal{N}$ *is a bit-serial algorithm that: (i) runs on an order-*d *omega machine, (ii) emulates the execution of* $\mathcal{N}$ *on any* d*-vector of* b*-bit integers, and (iii) receives (resp., produces) the* j*th component of the input (resp., output)* d*-vector at processor* (d, j), $0 \leq j < \mathrm{pow}(d)$.

We remark that the somewhat unnatural input-output convention used in the preceding definition (level d is used instead of level 0) is not essential. We could easily modify the results of this section to hold if the input and output are provided at level 0.

DEFINITION 11.2. *A depth-$(2 \cdot b)$ hypercubic d-network $\mathcal{N}$ is a-*pass*, $0 \le b \le a \le d$, if and only if the d-permutation $\hookrightarrow_d$ (resp., $\hookleftarrow_d$) is applied in the permutation phase of the first (resp., last) b levels of $\mathcal{N}$.*

LEMMA 11.1. *There is an $O(a+b)$-bit-step bit-serial omega emulation scheme for any coin-tossing a-pass word-size-b d-network.*

Proof. Straightforward. (Note that each of our five gate types can be implemented in a bit-serial fashion. For the "+" and "−" gates, such a bit-serial implementation requires that the inputs be provided most-significant-bit first.) □

DEFINITION 11.3. *A depth-$O(a)$ hypercubic d-network $\mathcal{N}$ is a-*multipass*, $0 \le a \le d$, if and only if $\mathcal{N}$ can be decomposed into an $O(1)$-length sequence of a-pass networks.*

LEMMA 11.2. *There is an $O(a+b)$-bit-step bit-serial omega emulation scheme for any coin-tossing a-multipass word-size-b d-network.*

Proof. This follows from a constant number of applications of Lemma 11.1. Note that only levels d through $d-a$ of the order-d omega machine are used by the emulation scheme. □

DEFINITION 11.4. *A hypercubic d-network $\mathcal{N}$ is (γ, a, k)-*geometric*, $\gamma \ge 0$, $0 \le a \le d$, $k \ge 0$, if and only if $\mathcal{N}$ can be decomposed into a length-k sequence of d-networks $\langle \mathcal{N}_i \rangle$ such that: (i) $0 \le \gamma \le 1$ and $\mathcal{N}_i$ is $\lfloor \gamma^{i+1} \cdot a \rfloor$-multipass, $0 \le i < k$, or (ii) $\gamma > 1$ and $\mathcal{N}_i$ is $\lfloor \gamma^{i-k} \cdot a \rfloor$-multipass, $0 \le i < k$.*

DEFINITION 11.5. *A (γ, a, k)-geometric d-network is *compact* if and only if*

$$\sum_{0 \le i < k} \left(\lfloor \gamma^{i+1} \cdot a \rfloor + 1 \right) \le d.$$

LEMMA 11.3. *There is an $O(b+d)$-bit-step bit-serial omega emulation scheme for any coin-tossing compact (γ, a, k)-geometric word-size-b d-network $\mathcal{N}$.*

Proof. We give the proof for the case $0 \le \gamma \le 1$. The case $\gamma > 1$ is similar. (Note that if we "reverse" a (γ, a, k)-geometric d-network, we obtain a $(1/\gamma, a, k)$-geometric d-network.)

Decompose the given d-network $\mathcal{N}$ into a sequence of networks $\langle \mathcal{N}_i \rangle$ as in Definition 11.4. Thus, network $\mathcal{N}_i$ is $(f(a,i)-1)$-multipass where

$$f(a,i) = \lfloor \gamma^{i+1} \cdot a \rfloor + 1,$$

$0 \le i < k$. Let $\mathcal{M}$ denote an order-d omega machine, and

$$g(i) = d - \sum_{0 \le j < i} f(a,j),$$

$0 \le i < k$. To obtain an efficient bit-serial omega emulation scheme for network $\mathcal{N}$, we use Lemma 11.2 to emulate $\mathcal{N}_i$ on the contiguous set of levels $A_i = \{g(i), g(i)-1, \ldots, g(i+1)+1\}$ of $\mathcal{M}$, $0 \le i < k$. (The input to $\mathcal{N}_i$ is provided at level $g(i)$. Hence, Lemma 11.2 implies that the output to network $\mathcal{N}_i$ is produced at level $g(i)$.) Note that: (i) the A_i's are well-defined (i.e., each is a set of level numbers in the range 0 to d) since $\mathcal{N}$ is compact, and (ii) the A_i's are pairwise disjoint. Property (ii) ensures that the emulation schemes associated with distinct $\mathcal{N}_i$'s do not interfere with one another.

It remains to prove that we can efficiently communicate the output of network $\mathcal{N}_i$, which is produced at level $g(i)$, to level $g(i+1)$, $0 \le i < k-1$. (Once the output of $\mathcal{N}_i$ reaches level $g(i+1)$, the emulation of $\mathcal{N}_{i+1}$ over the set of levels A_{i+1} can begin.) For example, it would suffice to efficiently map the value on wire j of level $g(i)$ to wire j of level $g(i+1)$, $0 \le j < \mathrm{pow}(d)$. Unfortunately, it is very expensive to implement such an identity d-permutation on the machine $\mathcal{M}$.

The key observation, however, is that we do not need to implement the identity d-permutation between levels $g(i)$ and $g(i+1)$. Because the $f(a,i)$'s form a nonincreasing sequence, $0 \le i < k$, there is no interaction between distinct $f(a,i+1)$-cubes in any d-network $\mathcal{N}_j$ such that $i+1 \le j < k$. Hence, it is sufficient to implement a d-permutation π such that: (i) level-$g(i)$ $f(a,i+1)$-cubes are mapped to level-$g(i+1)$ $f(a,i+1)$-cubes, and (ii) the identity $f(a,i+1)$-permutation is applied within each $f(a,i+1)$-cube.

We now prove that a d-permutation π of the desired form can be implemented in $O(f(a,i)+f(a,i+1))$ bit steps while using only levels $A_i \cup A_{i+1}$ of $\mathcal{M}$. (The reader may wonder why the preceding O-bound does not depend on b. This bound refers only to the "additional" number of bit steps required to implement the d-permutation π. Because our entire emulation scheme is pipelined bit-serially, we can account for the word size by adding $O(b)$ at the end of our analysis.) In particular, the d-permutation π that we apply from level $g(i)$ to level $g(i+1)$ satisfies

$$\pi(j) = j_{f(a,i)+f(a,i+1)-1} \cdots j_{f(a,i+1)} j_{d-1} \cdots j_{f(a,i)+f(a,i+1)} j_{f(a,i+1)-1} \cdots j_0,$$

$0 \le j < \mathrm{pow}(d)$. The d-permutation π can easily be implemented in $O(f(a,i)+f(a,i+1))$ additional bit steps as follows. First, within level $g(i)$, we apply the d-permutation π' defined by

$$\pi'(j) = j_{d-1} \cdots j_{f(a,i)+f(a,i+1)} j_{f(a,i+1)-1} \cdots j_0 j_{f(a,i)+f(a,i+1)-1} \cdots j_{f(a,i+1)},$$

$0 \le j < \mathrm{pow}(d)$. Note that π' belongs to $\Pi(d, f(a,i)+f(a,i+1))$. By Lemma 4.7, π' can be implemented in $O(f(a,i)+f(a,i+1))$ bit steps using only levels $A_i \cup A_{i+1}$ of $\mathcal{M}$. We now complete the implementation of d-permutation π by repeatedly unshuffling the data from level $g(i)$ to level $g(i+1)$. (This takes $f(a,i)$ bit steps and uses only the set of levels A_i.)

The total running time of our emulation scheme is easily seen to be

$$O\left(b + \sum_{0 \le i < k} f(a,i)\right) = O(b+d-g(k-1))$$

bit steps. However, one final detail remains to be addressed, since our emulation scheme leaves the output at level $g(k-1)$ instead of level d. We can move the output to level d by applying an appropriate fixed permutation π' from level $g(k-1)$ to level d. The entire machine $\mathcal{M}$ can be used for this purpose. Thus, the Beneš routing technique of Lemma 4.7 yields an immediate $O(b+d)$ bound. In fact, an $O(b+d-g(k-1))$ bound can easily be achieved by: (i) shuffling the data from level $g(k-1)$ to level d, and (ii) applying an appropriate permutation π'' in $\Pi(d, d-g(k-1))$.

Thus, the total running time of our revised emulation scheme is $O(b+d-g(k-1))$, which is $O(b+d)$ since $\mathcal{N}$ is compact. $\square$

THEOREM 11.1. *There is a polynomial-time uniform $O(b+d)$-bit-step bit-serial omega emulation of the family $\{\mathcal{N}_d\}$ of coin-tossing d-networks of Theorem 10.1.*

Proof. Examining the construction of Theorem 10.1, we find that $\mathcal{N}_d$ can be decomposed into: (a) a $(1, d, 1)$-geometric coin-tossing d-network, (b) a $(1/3, d, O(\lg d))$-geometric coin-tossing d-network, (c) a $(1, O(\sqrt{\lg d}), O(\sqrt{\lg d}))$-geometric deterministic d-network, (d) a $(3, d, O(\lg d))$-geometric coin-tossing d-network, and (e) a $(1, d, 1)$-geometric coin-tossing d-network. (Note that the constants 1/3 and 3 appearing above are somewhat arbitrarily chosen. The "slack" in our definitions creates a range of acceptable values.)

The coin-tossing d-networks of stages (a), (b), (d), and (e) are easily seen to be compact. Stage (c) corresponds to the use of bitonic sort to cut off the sorting recurrence at $O(\sqrt{\lg d})$-cubes. Note that we are free to choose the constant within the $O(\sqrt{\lg d})$ bound arbitrarily small. Hence, we can ensure that the deterministic d-network of stage (c) is also compact. The theorem then follows by Lemma 11.3. (The emulation is polynomial-time uniform since: (i) the family of coin-tossing networks of Theorem 10.1 is polynomial-time uniform, and (ii) the emulation scheme underlying Lemma 11.3 is polynomial-time uniform.) □

THEOREM 11.2. *For each function $\varepsilon(d) = \mathrm{pow}(-\mathrm{pow}(O(\sqrt{d})))$, there is a polynomial-time uniform $O(b + d)$-bit-step randomized bit-serial algorithm for an order-d omega machine that sorts any input d-vector of b-bit integers with probability at least $1-\varepsilon(d)$.*

Proof. This is an immediate consequence of Theorems 10.1 and 11.1. □

12. Concluding Remarks. We have defined the class of "hypercubic" sorting networks, and established a nearly logarithmic upper bound on the depth complexity of such networks. Of course, it would be very interesting to close the remaining gap. Given the techniques developed in this paper, the problem of constructing an optimal $O(\lg n)$-depth hypercubic sorting network has been reduced to the problem of constructing an $O(\lg n)$-depth comparator network that sorts a randomly chosen input permutation with probability at least $1 - 2^{-n^{\varepsilon}}$ for some constant $\varepsilon > 0$.

One unfortunate characteristic of our hypercubic sorting network construction is its lack of uniformity. In particular, no deterministic polynomial-time algorithm is known for generating the family of networks for which existence has been established. On the positive side, existence of a randomized polynomial-time generation algorithm is a straightforward consequence of our results.

While the multiplicative constant of approximately 7.44 established for the sorting network construction of §6 appears to be quite reasonable, the construction remains impractical. This is due to the fact that there is a trade-off between the value of the multiplicative constant and the success probability (the probability that a random input permutation is sorted by the network); for practical values of n, a significant increase in the constant is required in order to prove any reasonable success probability.

Acknowledgments. Thanks to Rostislav Caha, Don Coppersmith, Ming Kao, Sheralyn Listgarten, Yuan Ma, Bruce Maggs, and Torsten Suel for valuable discussions. We would also like to thank the anonymous referees who provided numerous valuable comments for improving the presentation of the paper.

REFERENCES

[1] W. Aiello, F. T. Leighton, B. Maggs, and M. Newman. Fast algorithms for bit-serial routing on a hypercube. *Mathematical Systems Theory*, 24:253–271, 1991.

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

[3] M. Ajtai, J. Komlós, and E. Szemerédi. Halvers and expanders. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 686–692, October 1992.
[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] V. E. Beneš. Optimal rearrangeable multistage connecting networks. *Bell System Technical Journal*, 43:1641–1656, 1964.
[6] H. Chernoff. A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–509, 1952.
[7] R. E. Cypher. Theoretical aspects of VLSI pin limitations. *SIAM Journal on Computing*, 22:58–63, 1993.
[8] R. E. Cypher and C. G. Plaxton. Techniques for shared key sorting. Technical Report RJ 7347, Computer Science Department, IBM Almaden Research Center, March 1990.
[9] R. E. Cypher and C. G. Plaxton. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. *Journal of Computer and System Sciences*, 47:501–548, 1993.
[10] W. Hoeffding. On the distribution of the number of successes in independent trials. *Annals of Mathematical Statistics*, 27:713–721, 1956.
[11] D. E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, Reading, MA, 1973.
[12] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*. Morgan-Kaufmann, San Mateo, CA, 1991.
[13] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. *Journal of Algorithms*, 17:157–205, 1994.
[14] 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.
[15] M. S. Paterson. Improved sorting networks with $O(\log N)$ depth. *Algorithmica*, 5:75–92, 1990.
[16] C. G. Plaxton. A hypercubic sorting network with nearly logarithmic depth. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 405–416, May 1992.
[17] C. G. Plaxton and T. Suel. A lower bound for sorting networks based on the shuffle permutation. *Mathematical Systems Theory*, 27:491–508, 1994.
[18] A. G. Ranade. How to emulate shared memory. *Journal of Computer and System Sciences*, 42:307–326, 1991.
[19] J. H. Reif and L. G. Valiant. A logarithmic time sort for linear size networks. *Journal of the ACM*, 34:60–76, 1987.
[20] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 263–277, May 1981.

A. Analysis of the Sorting Recurrence. Let c_1 be a positive constant, let a_0 and c_2 denote sufficiently large positive constants (to be determined), and consider the function $T(a)$, $a \geq 0$, defined by the following recurrence. Let $T(a) = O(1)$ for $a \leq a_0$, and

$$T(a) = \min_{(2+\lg a)\cdot\nu(b) \leq b \leq a} T(a - \lfloor \mu(b) \cdot b \rfloor) + T(b) + c_1 \cdot a \cdot \nu(a) \tag{15}$$

for $a > a_0$, where

$$\begin{aligned} \nu(d) &= c_2 \cdot \sqrt{\lg d}, \text{ and} \\ \mu(d) &= \mu_c - \frac{2}{\nu(d)}. \end{aligned}$$

Note that our choice of the function $\nu(d)$ satisfies the requirements of Lemma 9.2, since $\nu(d) = \omega(1)$ and $\nu(d) = o(d)$. Furthermore, the function $\mu(d)$ is defined as in Lemma 9.2. Hence, any upper bound for $T(a)$ is also an upper bound for the function $S(a)$ defined towards the end of §9. In this section, we prove that

$$T(a) = O(a \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a}) \cdot \lg a).$$

(The constant υ_c is defined in Equation (9).) We prove two technical lemmas before analyzing the recurrence of Equation (15).

LEMMA A.1. *For every pair of real numbers x and y such that $0 \leq x \leq y$, we have*

$$\sqrt{x \cdot y} - \sqrt{x \cdot y - x \cdot \sqrt{x \cdot y}} \geq x/2.$$

Proof. Examining the Taylor series expansion of $\sqrt{1-\varepsilon}$, we find that

$$\sqrt{1-\varepsilon} \leq 1 - \varepsilon/2$$

for all ε in $[0,1]$. Hence,

$$\begin{aligned} \sqrt{x \cdot y} - \sqrt{x \cdot y - x \cdot \sqrt{x \cdot y}} &= \sqrt{x \cdot y} - \sqrt{x \cdot y} \cdot \sqrt{1 - \sqrt{x/y}} \\ &\leq \sqrt{x \cdot y} - \sqrt{x \cdot y} \cdot \left(1 - \frac{\sqrt{x}}{2 \cdot \sqrt{y}}\right) \\ &= x/2. \end{aligned}$$

□

LEMMA A.2. *For any $a > a_0$, let*

$$a' = a - \lfloor \mu(b') \cdot b' \rfloor ,$$

where

$$b' = \left\lfloor \frac{a}{\mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a})} \right\rfloor .$$

For a sufficiently large choice of the constant a_0, the following bounds hold:

(i) $1 < a' \leq a - \frac{\mu(b') \cdot a}{\mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a})} + 2 < a$,

(ii) $\mu(b') \geq \mu_c - \frac{4}{c_2 \cdot \sqrt{\lg a}}$,

(iii) $2 \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a}) = o(a/\sqrt{\lg a})$,

(iv) $1 < (2 + \lg a) \cdot \nu(b') \leq b' < a$, *and*

(v) $\sqrt{\upsilon_c \cdot \lg a} - \sqrt{\upsilon_c \cdot \lg a - \upsilon_c \cdot \sqrt{\upsilon_c \cdot \lg a}} > \upsilon_c/2$.

Proof. Part (v) follows from Lemma A.1. The remaining inequalities follow from straightforward asymptotic estimates. □

THEOREM A.1. *There is a positive constant c_0 such that*

$$T(a) \leq c_0 \cdot a \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a}) \cdot \lg a.$$

for all $a > 1$.

Proof. We prove the claim of the lemma by induction on a. The claim is trivial for $1 < a \leq a_0$, since we are free to choose the constant c_0 arbitrarily large. Now fix $a > a_0$ and assume the claim holds for all smaller values of a. Define a' and b' as in Lemma A.2.

By Part (i) of Lemma A.2, we can apply the induction hypothesis to bound $T(a')$ since $1 < a' < a$. Using Parts (i), (ii), and (iii) of Lemma A.2, we find that

$$\begin{aligned} T(a') &\leq c_0 \cdot a' \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a'}) \cdot \lg a' \\ &\leq c_0 \cdot \left(a - \frac{\mu(b') \cdot a}{\mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a})} + 2\right) \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a}) \cdot \lg a \\ &\leq c_0 \cdot a \cdot \mathrm{pow}(\sqrt{\upsilon_c \cdot \lg a}) \cdot \lg a - c_0 \cdot \mu_c \cdot a \cdot \lg a + c_0 \cdot \left(\frac{4}{c_2} + o(1)\right) \cdot a \cdot \sqrt{\lg a}. \end{aligned}$$

By Part (iv) of Lemma A.2, we can apply the induction hypothesis to bound $T(b')$ since $1 < b' < a$. Using Part (v) of Lemma A.2 and Equation (9), we find that

$$\begin{aligned} T(b') &\leq c_0 \cdot b' \cdot \text{pow}(\sqrt{\upsilon_c \lg b'}) \cdot \lg b' \\ &\leq c_0 \cdot a \cdot \text{pow}\left(\sqrt{\upsilon_c \cdot \lg a - \upsilon_c \cdot \sqrt{\upsilon_c \cdot \lg a}} - \sqrt{\upsilon_c \cdot \lg a}\right) \cdot \left(\lg a - \sqrt{\upsilon_c \cdot \lg a}\right) \\ &\leq c_0 \cdot \mu_c \cdot a \cdot \lg a - c_0 \cdot \mu_c \cdot a \cdot \sqrt{\upsilon_c \cdot \lg a}. \end{aligned}$$

By Part (iv) of Lemma A.2, we can set $b = b'$ in the recurrence of Equation (15) since $(2 + \lg a) \cdot \nu(b') \leq b' < a$. Thus,

$$\begin{aligned} T(a) &\leq T(a') + T(b') + c_1 \cdot a \cdot \nu(a) \\ &\leq c_0 \cdot a \cdot \text{pow}(\sqrt{\upsilon_c \cdot \lg a}) \cdot \lg a \\ &\quad + \left[c_0 \cdot \left(\frac{4}{c_2} + o(1) - \mu_c \cdot \sqrt{\upsilon_c}\right) + c_1 \cdot c_2\right] \cdot a \cdot \sqrt{\lg a}. \end{aligned}$$

For a_0 sufficiently large, we can choose the constants c_0 and c_2 so that the coefficient of $a\sqrt{\lg a}$ is negative, yielding

$$T(a) \leq c_0 \cdot a \cdot \text{pow}(\sqrt{\upsilon_c \cdot \lg a}) \cdot \lg a,$$

as required. □

B. Locating the "Median" of the Binomial Distribution. The purpose of this appendix is to establish Theorem B.1, which is used in the proof of Lemma 5.15.

LEMMA B.1. *Let k and n be integers such that $0 \leq k \leq n$, p be a real number in $[0, 1]$ such that $np = k$, and X be a random variable drawn from $B(n, p)$ (i.e., the binomial distribution with parameters n and p). Let Y be a random variable corresponding to the number of successes in n independent Bernoulli trials with associated success probabilities p_i, $0 \leq i < n$, such that $\sum_{0 \leq i < n} p_i = k$. Then*

$$\begin{aligned} \Pr(X < k) &\geq \Pr(Y < k) \text{ and} \\ \Pr(X > k) &\geq \Pr(Y > k). \end{aligned}$$

Proof. These inequalities are established (in a more general form) by Hoeffding in [10, Theorem 4]. □

LEMMA B.2. *Let k, n, p, and X be as defined in Lemma B.1. If $0 \leq p \leq 1/2$, then let Y' and Y'' be random variables drawn from $B(2k, 1/2)$ and $B(n - 2k, 0)$, respectively. Otherwise, let Y' and Y'' be random variables drawn from $B(2(n - k), 1/2)$ and $B(2k - n, 1)$, respectively. Let $Y = Y' + Y''$. If*

$$\Pr(X = k) \geq \Pr(Y = k)/2$$

then

$$\min\{\Pr(X \leq k), \Pr(X \geq k)\} \geq 1/2.$$

Proof. By symmetry, $\Pr(Y \leq k) = \Pr(Y \geq k) = [1 + \Pr(Y = k)]/2$. The claimed inequality then follows easily using Lemma B.1. □

LEMMA B.3. *For each pair of integers k and n such that $0 \leq k \leq n$, let*

$$u_{k,n} = \binom{n}{k} \left(\frac{k}{n}\right)^k \left(1 - \frac{k}{n}\right)^{n-k}.$$

(If $n = k$, then $u_{k,n} = 1$.) Then $u_{k,n} \geq \frac{k^k}{e^k k!}$.

Proof. Fix $k \geq 0$, and let $v_n = u_{k,n}$ for $n \geq k$. It is sufficient to prove that the sequence $\langle v_n \rangle$ is nonincreasing for $n \geq k$, and that

$$\lim_{n \to \infty} v_n = \frac{k^k}{e^k k!}. \tag{16}$$

To see that the sequence $\langle v_n \rangle$ is nonincreasing for $n \geq k$, note that

$$\begin{aligned} \frac{v_{k+1}}{v_k} &= \left(\frac{k}{k+1}\right)^k \\ &\leq 1, \end{aligned}$$

and for $n > k$ we have

$$\begin{aligned} \frac{v_{n+1}}{v_n} &= \left(1 + \frac{1}{n-k}\right)^{n-k} \Big/ \left(1 + \frac{1}{n}\right)^n \\ &\leq 1. \end{aligned}$$

(The preceding inequality follows from the fact that the function $f : \mathbf{R} \to \mathbf{R}$ defined by $f(x) = (1 + \frac{1}{x})^x$ is strictly increasing for all $x \geq 1$.)

To verify Equation (16), note that

$$\begin{aligned} v_n &= \frac{n!}{k!(n-k)!} \cdot \frac{k^k}{n^k} \cdot \frac{(n-k)^{n-k}}{n^{n-k}} \\ &= \frac{k^k}{k!} \cdot \frac{n!}{(n-k)!} \cdot \frac{(n-k)^{n-k}}{n^n} \\ &\sim \frac{k^k}{k!} \left(1 - \frac{k}{n}\right)^n \\ &\sim \frac{k^k}{e^k k!}. \end{aligned}$$

□

LEMMA B.4. *For each nonnegative integer k, let*

$$w_k = \frac{k^{k+1} 2^{2k} k!}{(k+1) e^k (2k)!}.$$

Then the sequence $\langle w_k \rangle$ is nondecreasing for $k \geq 1$.

Proof. Note that

$$\frac{w_{k+1}}{w_k} = \frac{2(1 + \frac{1}{k})^k (k+1)^3}{ek(k+2)(2k+1)}$$

and

$$\lim_{k \to \infty} \frac{w_{k+1}}{w_k} = 1.$$

Thus, it is sufficient to prove that the function $f : \mathbf{R} \to \mathbf{R}$ defined by

$$f(x) = \frac{(1+\frac{1}{x})^x (x+1)^3}{x(x+2)(2x+1)}$$

is nonincreasing for $x \geq 1$. One may easily verify that

$$\frac{df(x)}{dx} = \frac{(x+1)^2(1+\frac{1}{x})^x}{x^2(x+2)^2(2x+1)^2} \cdot g(x)$$

where

$$g(x) = -x^2 - 6x - 2 + x(x+1)(x+2)(2x+1)[\ln(1+\tfrac{1}{x}) - \tfrac{1}{x+1}].$$

Hence, for $x > 0$, $\frac{df(x)}{dx} \leq 0$ if and only if $g(x) \leq 0$. For $x \geq 1$, we have $\ln(1+\frac{1}{x}) \leq \frac{1}{x} - \frac{1}{2x^2} + \frac{1}{3x^3}$, and hence

$$\begin{aligned} g(x) &\leq -x^2 - 6x - 2 + x(x+1)(x+2)(2x+1)(\tfrac{1}{x(x+1)} - \tfrac{1}{2x^2} + \tfrac{1}{3x^3}) \\ &= -\tfrac{23x}{6} - \tfrac{7}{6} + \tfrac{4}{3x} + \tfrac{2}{3x^2} \\ &< 0. \end{aligned}$$

□

LEMMA B.5. *For all nonnegative integers k, we have*

$$\frac{k^k}{e^k k!} > \binom{2k}{k} 2^{-2k-1}.$$

Proof. It is easy to verify that the claim holds for $0 \leq k \leq 2$. For $k \geq 3$, consider the sequence $\langle w_k \rangle$ defined in Lemma B.4. By Lemma B.4, $w_k \geq w_3 \approx 0.538 > 1/2$ for $k \geq 3$. Hence

$$\begin{aligned} \frac{k^k}{e^k k!} &> \frac{k+1}{k}\binom{2k}{k} 2^{-2k-1} \\ &> \binom{2k}{k} 2^{-2k-1}. \end{aligned}$$

□

LEMMA B.6. *Let k, n, p, X, and Y be as defined in Lemma B.2. Then*

$$\Pr(X = k) > \Pr(Y = k)/2.$$

Proof. If $0 \leq p \leq 1/2$, then

$$\begin{aligned} \Pr(X = k) &= \binom{n}{k}\left(\frac{k}{n}\right)^k \left(1 - \frac{k}{n}\right)^{n-k} \\ &\geq \frac{k^k}{e^k k!} \\ &> \binom{2k}{k} 2^{-2k-1} \\ &= \Pr(Y = k)/2, \end{aligned}$$

where the two inequalities follow from Lemmas B.3 and B.5, respectively.

Similarly, if $1/2 < p \leq 1$, we have

$$\begin{aligned}
\Pr(X = k) &= \binom{n}{k} \left(\frac{k}{n}\right)^k \left(1 - \frac{k}{n}\right)^{n-k} \\
&= \binom{n}{n-k} \left(\frac{n-k}{n}\right)^{n-k} \left(1 - \frac{n-k}{n}\right)^{n-(n-k)} \\
&\geq \frac{(n-k)^{n-k}}{e^{n-k}(n-k)!} \\
&> \binom{2(n-k)}{n-k} 2^{-2(n-k)-1} \\
&= \Pr(Y = k)/2.
\end{aligned}$$

□

LEMMA B.7. *Let k, n, p, and X be as defined in Lemma B.1. Then*

$$\min\{\Pr(X \leq k), \Pr(X \geq k)\} \geq 1/2.$$

Proof. Immediate from Lemmas B.2 and B.6. □

THEOREM B.1. *Let n be a nonnegative integer, p be a real number in $[0,1]$, and X be a random variable drawn from $B(n,p)$. Then*

$$\min\{\Pr(X \geq \lfloor np \rfloor), \Pr(X \leq \lceil np \rceil)\} \geq 1/2.$$

Proof. Define real numbers p^- and p^+ so that $np^- = \lfloor np \rfloor$ and $np^+ = \lceil np \rceil$. Let X^- (resp., X^+) denote a random variable drawn from $B(n,p^-)$ (resp., $B(n,p^+)$). Note that for all real numbers x, we have

$$\begin{aligned}
\Pr(X \geq x) &\geq \Pr(X^- \geq x) \text{ and} \\
\Pr(X \leq x) &\geq \Pr(X^+ \leq x).
\end{aligned}$$

Combining these inequalities with the bound of Lemma B.7, we obtain

$$\begin{aligned}
&\Pr(X \geq \lfloor np \rfloor) \geq \Pr(X^- \geq \lfloor np \rfloor) \geq 1/2 \text{ and} \\
&\Pr(X \leq \lceil np \rceil) \geq \Pr(X^+ \leq \lceil np \rceil) \geq 1/2.
\end{aligned}$$

□