A Hypercubic Sorting Network with Nearly Logarithmic Depth

C. Greg Plaxton*
Department of Computer Science
University of Texas at Austin

Abstract

A natural class of "hypercubic" sorting networks is defined. The regular structure of these sorting networks allows for elegant and efficient implementations on any of the so-called hypercubic networks (e.g., the hypercube, shuffle-exchange, butterfly, and cube-connected cycles). This class of sorting networks contains Batcher's $O(\lg^2 n)$-depth bitonic sort, but not the $O(\lg n)$-depth sorting network of Ajtai, Komlós, and Szemerédi. In fact, no $o(\lg^2 n)$-depth compare-interchange sort was previously known for any of the hypercubic networks. In this paper, we prove the existence of a family of $2^{O(\sqrt{\lg\lg n})}\lg n$-depth hypercubic sorting networks. Note that this depth is $o(\lg^{1+\epsilon} n)$ for any constant $\epsilon > 0$.

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 inputs to the network may be thought of as arbitrary integer values. The two outputs of each comparator gate are labelled MAX and MIN, respectively, while the two inputs are not labeled. On input x and y, a comparator gate emits $\max\{x,y\}$ on its MAX output and $\min\{x,y\}$ on its MIN output. It is straightforward to prove (by induction on the depth of the network) that any comparator network induces some permutation of the input (an integer n-tuple) at the n outputs. We say that a given comparator network *sorts* a particular n-tuple vector if the value emitted at output i is less than or equal to the value emitted at output $i+1$, $0 \le i < n-1$.

An *n-input sorting network* is an n-input comparator network that sorts every possible input vector. It is straightforward to prove that any n-input comparator network that sorts the $n!$ permutations of $[n] \overset{\text{def}}{=} \{0,\ldots,n-1\}$ is an n-input sorting network. In fact, any n-input comparator network that sorts the 2^n n-tuples of $\{0,1\}^n$ is an n-input sorting network. The latter observation is known as the 0-1 principle for sorting networks [6].

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 [2]. For small values of n, the depth of bitonic sort either matches or is very

*This research was supported by NSF Research Initiation Award CCR-9111591, and Texas Advanced Research Program (TARP) Award #003658480.

close to matching that of the best constructions known (a very limited number of which are known to be optimal) [6]. 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 (posed as an open problem): "Prove that the asymptotic value of $\hat{S}(n)$ is not $O(n \log n)$," where $\hat{S}(n)$ denotes the minimal size (number of comparator gates) of an n-input sorting network of *any* depth. One source of the difficulty of this particular exercise was subsequently revealed by Ajtai, Komlós, and Szemerédi [1], who provided an optimal $O(\lg n)$-depth sorting network construction (hereinafter referred to 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 by 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 (a hypercubic network) requires $\Omega(\lg^2 n)$ time [5]. 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 network that is also useful for performing operations other than sorting.

In [7], Leighton and Plaxton pursue a novel approach towards circumventing the two shortcomings of the AKS network mentioned above. They consider the construction of small-depth "probabilistic" sorting networks that sort most, but not all, of the $n!$ possible permutations. Their main result is an $O(\lg n)$-depth comparator network (hereinafter referred to as the "LP probabilistic sorting network") that sorts all but $\epsilon n!$ of the $n!$ possible input permutations, where

$$\epsilon = 2^{-2^{c\sqrt{\lg n}}}$$

for any constant $c > 0$. It should be emphasized that the LP probabilistic sorting network is in fact a standard, deterministic comparator network. The use of the term "probabilistic" is motivated by the fact that a device that sorts most permutations will sort a randomly chosen permutation with high probability. At the expense of failing to sort a small fraction of the possible input permutations, the LP probabilistic sorting network provides two advantages over the AKS construction: (i) the multiplicative constant hidden by the O-notation is significantly smaller, and (ii) the "regular" construction permits an efficient implementation on any of the *hypercubic networks* (e.g., the hypercube, shuffle-exchange, butterfly, and cube-connected cycles).

The goal of the present paper is to demonstrate that the LP probabilistic sorting network can be used as a building block for a nearly logarithmic depth family of deterministic sorting networks. Furthermore, these networks can be mapped efficiently (with a small constant factor slowdown) to any of the hypercubic networks. Before elaborating on this result in greater detail, we will describe a more formal framework for approaching the notion of "network regularity" alluded to above.

An interesting alternative view of sorting networks is given by the following formulation. Define an n-input comparator network of depth d as a sequence of d pairs (π_i, v_i), $0 \le i < d$, where each π_i denotes a permutation of $[n]$, and each v_i denotes a bit vector of length $\lfloor n/2 \rfloor$. Imagine the input to the network (an integer n-tuple, as above) as being arranged in a set of n registers numbered from 0 to $n-1$. Define $\lfloor n/2 \rfloor$ disjoint pairs of registers by pairing register $2i$ with register $2i+1$, $0 \le i < \lfloor n/2 \rfloor$. The network runs in d steps, where the action of the ith step is determined by the pair (π_i, v_i), $0 \le i < d$. At the beginning of each step, every register will contain a single value. To perform the ith step, the contents of the n registers are first permuted according to permutation π_i. After applying this permutation, the following $\lfloor n/2 \rfloor$ operations are performed, one corresponding to each bit in the vector v_i. [The order in which these operations is performed can easily be seen to be immaterial.] The operation corresponding to $b \overset{\text{def}}{=} v_i(j)$, the jth bit of v_i, $0 \leq j < \lfloor n/2 \rfloor$, may be described as follows. If $b = 0$, then no action is performed. If $b = 1$, then the values in the jth pair of registers are interchanged if the value stored in register $2j$ is greater than the value stored in register $2j+1$. Such a comparator network is called a sorting network if and only if every possible input permutation is mapped to the identity permutation.

It is straightforward to see that the two definitions of sorting networks provided thusfar are equivalent in power: A sorting network $\mathcal{N}$ defined in one model is isomorphic to an equal-depth network $\mathcal{N}'$ defined in the other model. The appeal of the definition offered in the preceding paragraph is that it suggests an approach towards formalizing the notion of network regularity. For instance, one might ask whether it is possible to construct a small-depth sorting network in which all of the π_i's are equal to the same fixed permutation π. We will in fact be interested in considering various restrictions on the π_i's, and because of this, it will be natural to alter the definition of the v_i's slightly. Following the notation of Knuth's Exercise 5.3.4.47, we define v_i as a $\lfloor n/2 \rfloor$-tuple of values drawn from the set $\{0, 1, -, +\}$. The operation corresponding to the jth such value is as follows.

"0": Do nothing.

"1": Interchange the values stored in registers $2j$ and $2j+1$. Note: Knuth does not actually consider this operation.

"+": Compare the values in the jth pair of registers, and interchange them if the value stored in register $2j$ is greater than the value stored in register $2j+1$.

"−": Compare the values in the jth pair of registers, and interchange them if the value stored in register $2j$ is less than the value stored in register $2j+1$.

We will refer to any comparator (resp., sorting) network presented in this format as a *(π_i, v_i)-type comparator (resp., sorting) network.* Given that every sorting network is also a (π_i, v_i)-type sorting network, this definition will only be of interest in the context of some associated restriction on either the π_i's or the v_i's.

For $n = 2^d$ where d is a nonnegative integer, the shuffle permutation π_{sh} is defined as follows. If $i_{d-1} \cdots i_0$ denotes the binary representation of some integer i, $0 \leq i < 2^d$, we have

$$\pi_{\text{sh}}(i_{d-1} \cdots i_0) = i_{d-2} \cdots i_0 i_{d-1}.$$

The "unshuffle" permutation is simply the inverse of the permutation π_{sh} and will be denoted π_{sh}^{-1}. Knuth's Exercise 5.3.4.47 (posed as an open problem) may be viewed as asking for the depth complexity of (π_i, v_i)-type sorting networks in which every permutation π_i is equal to π_{sh}. Batcher's bitonic sort provides an $O(\lg^2 n)$ upper bound for this problem, and in a recent paper, Plaxton and Suel [9] have established an $\Omega(\lg^2 n / \lg\lg n)$ lower bound. Of course, the same lower bound applies if all of the π_i's are set to π_{sh}^{-1}.

From a practical point of view, it may be unnecessarily restrictive to consider (π_i, v_i)-type sorting networks in which either every permutation π_i is equal to π_{sh}, or every permutation π_i is equal to π_{sh}^{-1}. For instance, parallel computers based on hypercubic networks do not typically limit the programmer to strict "ascend" (or strict "descend") algorithms but will allow efficient implementation of any "ascend/descend" algorithm. This observation motivates the following definition. A *hypercubic comparator (resp., sorting) network* is defined as a (π_i, v_i)-type comparator (resp., sorting) network in which each permutation π_i belongs to the set $\{\pi_{\text{sh}}, \pi_{\text{sh}}^{-1}\}$. In view of the $\Omega(\lg^2 n / \lg\lg n)$ lower bound of Plaxton and Suel, the main result of the present paper is perhaps unexpected, namely, that there exist hypercubic sorting networks of depth

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

Note that this depth is $o(\lg^{1+\epsilon} n)$ for any constant $\epsilon > 0$. A more precise form of the upper bound is given in Section 5.

The remainder of the paper is organized as follows. Section 2 contains various additional definitions. Section 3 reviews the sorting properties guaranteed by the LP construction; as mentioned above, the LP probabilistic sorting network construction provides the basic building block for the family of networks defined in the present paper. Section 4 describes how to construct a high-order merging network from a comparator network that sorts most input permutations. Section 5 makes use of the high-order merging network to develop a somewhat unusual recurrence for the depth complexity of sorting. The analysis of this recurrence is presented in Appendix A. Section 6 offers some concluding remarks.

2 Additional definitions

Let $\mathbf{Z}^n$ denote the set of all integer n-tuples, and let $\Pi(n)$ denote the set of $n!$ permutations over $[n] \overset{\text{def}}{=} \{0, \ldots, n-1\}$.

A 0-1 permutation of length n is an n-tuple over $\{0,1\}$. Thus $\{0,1\}^n$ denotes the set of 2^n 0-1 permutations.

The input-output behavior of any particular sorting network corresponds to a mapping from $\mathbf{Z}^n$ to $\mathbf{Z}^n$. If we restrict our attention to inputs in $\Pi(n)$, the mapping is from $\Pi(n)$ to $\Pi(n)$. Similarly, by restricting the input to $\{0,1\}^n$ we obtain a mapping from $\{0,1\}^n$ to $\{0,1\}^n$.

Given a sorting network $\mathcal{N}$, we define $Sort(\mathcal{N})$ as the set of all integer n-tuples sorted by $\mathcal{N}$.

Let $S(d)$ denote the optimal depth of a 2^d-input hypercubic sorting network.

Let $\mathcal{M}(a,b)$ denote the set of all integer 2^{a+b}-tuples $\vec{x} = (x_0, \ldots, x_{2^{a+b}-1})$ such that the components of $\vec{x}$ are sorted in 2^a blocks of size 2^b. In other words, we require that $x_{i2^b+j} \leq x_{i2^b+j+1}$ for $0 \leq i < 2^a$ and $0 \leq j < 2^b - 1$.

Let $M(a,b)$ denote the optimal depth of a 2^{a+b}-input hypercubic comparator network $\mathcal{N}$ such that $\mathcal{M}(a,b) \subseteq Sort(\mathcal{N})$.

Let π denote an arbitrary permutation in $\Pi(n)$ and let k denote an integer in $[n+1]$. We define the kth 0-1 permutation corresponding to π, denoted $\pi_k^{0\text{-}1}$, as follows

$$\pi_k^{0\text{-}1}(i) = \begin{cases} 0 & \text{if } 0 \leq \pi(i) < k, \\ 1 & \text{if } k \leq \pi(i) < n. \end{cases}$$

Note that $\pi_k^{0\text{-}1}$ contains k 0's and $(n-k)$ 1's. For any permutation π in $\Pi(n)$, let $\Gamma(\pi)$ denote the set of $(n+1)$ 0-1 permutations corresponding to π. Formally, we have

$$\Gamma(\pi) = \bigcup_{0 \leq k \leq n} \pi_k^{0\text{-}1}.$$

For any permutation π in $\Pi(2^d)$, let $\mathcal{N}_\pi$ denote the 2^d-input hypercubic comparator network of depth $2d-1$ based on Beneš permutation routing [3]. Note that of the four operations $\{0, 1, +, -\}$ defined in the introduction, only $\{0,1\}$ are used in $\mathcal{N}_\pi$.

Several variable names are used throughout the paper. The constant δ^*, the function δ, and the function f (with associated constant c) are defined in Section 3. The function k_0 and the constant σ^* are defined in Section 5. The function σ is defined in Appendix A.

The phrase "sufficiently large" should be interpreted as "larger than some appropriately chosen positive constant."

3 Sorting most inputs

Throughout this section as well as the remainder of the paper, let

$$\begin{aligned} \delta^* &\overset{\text{def}}{=} \lg(4 - 2\sqrt{2}) \approx 0.228, \\ f(d) &\overset{\text{def}}{=} \frac{2 \lg^c d}{\delta^*}, \text{ and} \\ \delta &\overset{\text{def}}{=} \delta^* - \frac{1}{f(d)}, \end{aligned}$$

where c denotes some nonnegative real constant. The source of the constant δ^* will become evident in the proof of Lemma 3.1 below. Note that δ is not a constant, but a function of d. However, for $d \geq 2$, δ is bounded from above and below by constants since $\delta^*/2 \leq \delta \leq \delta^*$.

Theorem 1 below does not appear in [7], but it can be proven relatively easily given the techniques developed in that paper. Note that the theorem expresses a trade-off between the sorting power of a network and its depth; this trade-off is controlled by the choices of k and f. In order to focus on the essence of the claim without becoming overly distracted by the technical details of this trade-off, the reader may initially prefer to think of f as a (large) constant function. In Section 5, we will find that our best upper bound on the depth of hypercubic sorting networks is actually obtained by applying Theorem 1 with

$$\begin{aligned} k &= \left\lfloor d/2^{\sqrt{\alpha \lg d}} \right\rfloor, \text{ and} \\ f(d) &= \Theta(\sqrt{\lg d}). \end{aligned}$$

Theorem 1 Let d, k, and n denote integers such that $n = 2^d$ and $0 \le k < d$. There exists an n-input hypercubic comparator network $\mathcal{N}$ of depth

$$S(k) + \Theta(d + (d-k)f(d))$$

such that

$$|Sort(\mathcal{N}) \cap \Pi(n)| \ge (1 - O(n^3 2^{-2^{\delta k}}))n!.$$

□

We begin our progress towards a proof of Theorem 1 by considering the following essential lemma adapted from the "butterfly tournament" analysis of [7].

Lemma 3.1 Let $n = 2^d$ for some nonnegative integer d, let $\gamma = 1 - \frac{2}{3f(d)}$, and let $X = [n]$. Then there exists a fixed permutation π of X and a fixed subset Y of X such that $|Y| = O(n^\gamma)$ and the following statement holds true with probability at least $1 - O(n^{3/2} 2^{-2^{\delta d}})$: If a random input permutation is applied to an n-input butterfly comparator network then output i will receive a value in the range $[\pi(i) - O(n^\gamma), \pi(i) + O(n^\gamma)]$ for all i in $X \setminus Y$.

Proof: This lemma is adapted from Theorem 1 of [7], with two main changes. First, while the original theorem set γ to a particular constant (approximately 0.822) less than 1, it is now being set to $1 - \frac{2}{3f(d)}$. Second, the probability of failure explicitly stated above is significantly smaller than the bound implicit in the statement of the original theorem. These changes do not result from any new techniques for analyzing the behavior of butterfly comparator networks. Rather, they reflect a trade-off between the value of γ and the probability of failure. For the application described in [7] (a $7.44 \lg n$-depth network that sorts a randomly chosen input permutation with very high probability), the best trade-off is obtained by setting γ to the smallest possible value for which the probability of failure is less than $2^{-2^{\epsilon d}}$ for some constant $\epsilon > 0$. For the present application, we prefer to allow γ to approach 1 in order to obtain a very small probability of failure.

We now indicate how to adapt the arguments of [7], which led to a "low" γ, "high" probability of failure result, to obtain the desired "high" γ, "low" probability of failure result. [Unfortunately, the proof provided in [7] is rather lengthy, and so it will not be reproduced in its entirety.] We first modify the statement of Lemma 3.4 of [7]. The revised claim is that at most $n^{1-1/f(d)}$ of the $h_\alpha(p,q)$'s can exceed $n^{-\delta^* + \frac{1}{\lambda f(d)}}$. If this bound did not hold then we would have

$$\begin{aligned} H_\lambda(d,p,q) &> n^{1-1/f(d)} n^{-\delta^*\lambda + 1/f(d)} \\ &= n^{1-\delta^*\lambda}. \end{aligned}$$

On the other hand, setting $\lambda = 3$ we find that $r_3^* = (10 + 7\sqrt{2})/16$ (as discussed in [7]), and hence

$$\begin{aligned} H_3(d,p,q) &\le (r_3^*)^d \\ &= n^{\lg(10+7\sqrt{2})-4} \\ &= n^{1-3\lg(4-2\sqrt{2})}. \end{aligned}$$

Note that the constant δ^* has been defined in such a way that the preceding upper and lower bounds on $H_3(d,p,q)$ yield the desired contradiction. No other choice of the constant λ would allow δ^* to be set to a larger value, and so we may conclude that the stated value of δ^* is "best possible" with regard to this particular approach. On the other hand, the approach itself almost certainly does not provide a tight bound on the actual performance of the butterfly comparator network. In other words, it is likely that a tight analysis would result in a slightly higher value for the constant δ^* than $\lg(4 - 2\sqrt{2})$.

Given the bound established in the preceding paragraph, we can now define the set Y introduced in the statement of the lemma. Namely, we can define Y as the set of at most $n^{1-1/f(d)} \leq n^\gamma$ outputs α for which $h_\alpha(2^{-n^\delta}, 1 - 2^{-n^\delta})$ exceeds $n^{-\delta^* + \frac{1}{3f(d)}}$. For each output α in $X \setminus Y$, we can now argue as in [7] that

$$\begin{aligned} v_\alpha - u_\alpha &\leq 2n^\delta n^{-\delta^* + \frac{1}{3f(d)}} \\ &= O\left(n^{-\frac{2}{3f(d)}}\right). \end{aligned}$$

In what follows, let i denote the integer corresponding to the binary string α. By definition, we have $g_i(u_\alpha) = 2^{-n^\delta}$ and $g_i(v_\alpha) = 1 - 2^{-n^\delta}$. Thus, the reasoning used to prove Lemma 3.2 of [7] gives $f_i(\lfloor u_\alpha n \rfloor) = O(\sqrt{n}2^{-n^\delta})$ and $f_i(\lfloor v_\alpha n \rfloor) = 1 - O(\sqrt{n}2^{-n^\delta})$. Lemma 3.1 of [7] implies that output i will have rank k in the range $\lfloor u_\alpha n \rfloor \leq k < \lceil v_\alpha n \rceil$ with probability $1 - O(\sqrt{n}2^{-n^\delta})$. Since $|X \setminus Y| \leq |X| = n$, these bounds will hold for all binary strings α in $X \setminus Y$ simultaneously with probability $1 - O(n^{3/2}2^{-n^\delta})$. Following [7], the desired permutation π can now be constructed by sorting the outputs according to the p_α's. This completes the proof of Lemma 3.1. □

With the preceding lemma in hand, we are now able to sketch the proof of Theorem 1. As in the proof of Lemma 3.1, we will not reproduce in detail those arguments which are treated at length in [7], but will focus our discussion on the additional insights needed to establish Theorem 1.

Given a comparator network $\mathcal{N}$ and an input permutation π, we say that $\mathcal{N}$ *sorts* π *to within* Δ *positions* if every output i receives a value in the range $i - \Delta$ to $i + \Delta$. Let $n = 2^d$, and let $\mathcal{N}$ denote an n-input comparator network that (on a random input permutation) sorts all but a known set of n^γ outputs to within n^γ positions with probability at least $1 - \epsilon$, for real values $0 < \gamma < 1$ and $\epsilon \geq 0$. In [7], it is shown that an n-input comparator network $\mathcal{N}'$ consisting of:

(i) the network $\mathcal{N}$, followed by

(ii) some fixed permutation π_0 (for a hypercubic construction, this can be implemented with the network $\mathcal{N}_{\pi_0}$), followed by

(iii) a set of disjoint binary tree insertion networks applied to low-order subcubes of the output, followed by

(iv) some fixed permutation π_1 (for a hypercubic construction, the network $\mathcal{N}_{\pi_1}$ can be used),

will sort a randomly chosen input permutation to within $O(n^\gamma)$ positions with probability at least $1 - \epsilon$. The depth of the network $\mathcal{N}'$ exceeds that of network $\mathcal{N}$ by $\Theta(d)$. Furthermore, if $\mathcal{N}$ is a hypercubic comparator network, then $\mathcal{N}'$ can be implemented as a hypercubic comparator network with an additional depth that remains $\Theta(d)$. Thus, Lemma 3.1 implies the existence of an n-input, $\Theta(d)$-depth hypercubic comparator network $\mathcal{N}_d^*$ that sorts a randomly chosen input permutation to within

$$O(n^{1-\frac{2}{3f(d)}})$$

positions with probability at least

$$1 - O\left(n^{3/2}2^{-2^{\delta d}}\right).$$

Having defined the family of hypercubic comparator networks $\mathcal{N}_d^*$, a simple variant of the LP probabilistic sorting network construction may be described as follows. The construction will be applied only for sufficiently large values of d (for $d = O(1)$, we can sort in constant depth).

1. Apply the $\Theta(d)$-depth network $\mathcal{N}_d^*$. The output is now sorted to within n^γ positions for some real value $\gamma = 1 - \frac{2}{3f(d)} + O(1/d)$.

2. Apply some fixed permutation π (for a hypercubic construction, the network $\mathcal{N}_\pi$ can be used).

3. "Sort" low-order subcubes of dimension $\lceil \gamma d \rceil$ recursively unless the depth of recursion is such that the dimension of the subcubes to be sorted is less than or equal to k, in which case we apply an optimal-depth hypercubic sorting network of depth $S(k)$.

4. Merge adjacent pairs of subcubes using two sets of bitonic merge operations.

The last step is needed in order to take care of boundary effects across subcubes. The overall depth of the construction is easily proven to be $S(k) + \Theta(d + (d-k)f(d))$, and it is straightforward to prove that the same asymptotic bound is achievable in the hypercubic comparator network model.

To prove Theorem 1, we also need to provide an upper bound on the probability of failure of the probabilistic sorting network given by the preceding construction (i.e., the probability that the network fails to sort a randomly chosen input permutation). An easy upper bound is given by the sum of the individual probabilities of failure of the $\mathcal{N}_i^*$-type subnetworks (with various values of i, $k < i \leq d$) from which it is composed. The number of such subnetworks is clearly $O(n^{3/2})$ (a crude bound), since they are disjoint and the total number of gates in the network is $O(ndf(d)) = O(n^{3/2})$. Furthermore, the probability of failure of each subnetwork (i.e., the probability that it fails to sort every output to within $2^{\gamma k}$ positions) is

$$O\left(n^{3/2}2^{-2^{\delta k}}\right).$$

Hence, the overall probability of failure is

$$O\left(n^3 2^{-2^{\delta k}}\right),$$

and the proof of Theorem 1 is complete.

4 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 $n/2$ requires $\Omega(\lg n)$ depth, one cannot hope to obtain a $o(\lg^2 n)$-depth sorting network (hypercubic or otherwise) by repeated 2-way merging. This section describes how to use a comparator network $\mathcal{N}$ that sorts most inputs to construct a high-order merging network, that is, a k-way merging network for some $k \gg 2$. A similar technique has recently been used by Ajtai, Komlós, and Szemerédi [4] as part of an improved version of their original sorting network construction. The multiplicative constant associated with the new construction is significantly lower than the constant established by Paterson [8].

The following lemma represents a slight generalization of the 0-1 principle cited in Section 1.

Lemma 4.1 Let $\mathcal{N}$ denote an n-input comparator network. Let $X \subseteq \{0,1\}^n$ and let

$$Y = \{\pi \in \Pi(n) \mid \Gamma(\pi) \subseteq X\}.$$

Then $X \subseteq \mathit{Sort}(\mathcal{N})$ if and only if $Y \subseteq \mathit{Sort}(\mathcal{N})$.

Proof: By a straightforward extension of the proof of the 0-1 principle, which corresponds to the case where $X = \{0,1\}^n$. □

The following special case of Lemma 4.1 will prove to be useful.

Corollary 4.1.1 Let a and b denote two non-negative integers, let $n = 2^{a+b}$, and let $\mathcal{N}$ denote an arbitrary n-input comparator network. Then $\mathcal{M}(a,b) \cap \{0,1\}^n \subseteq \mathit{Sort}(\mathcal{N})$ if and only if $\mathcal{M}(a,b) \cap \Pi(n) \subseteq \mathit{Sort}(\mathcal{N})$.

In the sequence of lemmas to follow, we will make use of a network $\mathcal{N}$ that sorts most input permutations in order to construct a network $\mathcal{N}'$ that sorts most or all of a fixed set of permutations. In each case, the depth of network $\mathcal{N}'$ only exceeds the depth of network $\mathcal{N}$ by the depth required to implement an arbitrary fixed permutation. In the classical sorting network model, no additional depth is needed to implement a fixed permutation. On a 2^d-input hypercubic sorting network, an arbitrary permutation can be implemented in depth $2d-1$ using Beneš routing as discussed in Section 2. While this disparity leaves open the possibility that our network constructions will be significantly more efficient (i.e., by more than a constant factor) in the classical comparator network model than in the hypercubic comparator network model, such is not the case. In fact, the total depth devoted to routing fixed permutations will account for only a constant fraction of the overall depth of our hypercubic comparator networks. Furthermore, the method by which fixed permutations are implemented represents the sole difference between our constructions in the classical model and the corresponding constructions in the hypercubic model.

Lemma 4.2 Let $\mathcal{N}$ denote an n-input comparator network such that $|Sort(\mathcal{N}) \cap \Pi(n)| \geq (1-\epsilon)n!$, $0 \leq \epsilon \leq 1$. Then for every set $X \subseteq \Pi(n)$ there exists a permutation π in $\Pi(n)$ such that the network $\mathcal{N}'$ consisting of $\mathcal{N}_\pi$ followed by $\mathcal{N}$ satisfies

$$|Sort(\mathcal{N}') \cap X| \geq (1-\epsilon)|X|.$$

The preceding claim also holds for $X \subseteq \{0,1\}^n$.

Proof: We first consider the case $X \subseteq \Pi(n)$. Construct an undirected bipartite graph (U, V, E) as follows. Let the vertices of U be in one-to-one correspondence with the elements of X, and let the vertices of V be in one-to-one correspondence with the elements of $\Pi(n)$. Thus $|U| = |X|$ and $|V| = n!$. A vertex u in U is connected to a vertex v in V if and only if the permutation u is sorted by the circuit $\mathcal{N}'$ that results if π is set to v. The degree of every vertex in U is equal to $|Sort(\mathcal{N}) \cap \Pi(n)| \geq (1-\epsilon)n!$, and so the sum of the degrees of the vertices in U is at least $(1-\epsilon)|X|n!$. This sum is identical to that attained over V, and so some vertex v in V must have degree at least $(1-\epsilon)|X|$. Set $\pi = v$.

Now consider the case $X \subseteq \{0,1\}^n$. Construct a set $Y \subseteq \Pi(n)$, $|Y| \leq |X|$, by choosing for each 0-1 permutation $\pi^{0\text{-}1}$ in X a permutation π such that $\pi^{0\text{-}1}$ belongs to $\Gamma(\pi)$. Now apply the preceding argument to the set Y, along with Lemma 4.1. □

We will only make use of the following special case of Lemma 4.2.

Corollary 4.2.1 Let $\mathcal{N}$ denote an n-input comparator network such that $|Sort(\mathcal{N}) \cap \Pi(n)| \geq (1-\epsilon)n!$, $0 \leq \epsilon \leq 1$. Then for every set $X \subseteq \Pi(n)$ with $|X| < 1/\epsilon$ there exists a permutation π in $\Pi(n)$ such that the network $\mathcal{N}'$ consisting of $\mathcal{N}_\pi$ followed by $\mathcal{N}$ satisfies

$$|X| \subseteq Sort(\mathcal{N}').$$

The preceding claim also holds for $X \subseteq \{0,1\}^n$.

Proof: Immediate from Lemma 4.2 and the observation that $|Sort(\mathcal{N}') \cap X|$ must be an integer. □

Lemma 4.3 Let a and b denote two nonnegative integers, let $n = 2^{a+b}$, and let $\mathcal{N}$ denote an n-input comparator network such that $|Sort(\mathcal{N}) \cap \Pi(n)| \geq (1-\epsilon)n!$, $0 \leq \epsilon \leq 1$. Further assume that

$$(2^b + 1)^{2^a} < 1/\epsilon.$$

Then there exists a permutation π in $\Pi(n)$ such that the network $\mathcal{N}'$ consisting of $\mathcal{N}_\pi$ followed by $\mathcal{N}$ satisfies

$$\mathcal{M}(a,b) \cap \Pi(n) \quad \subseteq \quad Sort(\mathcal{N}').$$

Proof: By Corollary 4.1.1 it is sufficient to prove the existence of a permutation π in $\Pi(n)$ such that

$$\mathcal{M}(a,b) \cap \{0,1\}^n \quad \subseteq \quad Sort(\mathcal{N}').$$

Note that

$$|\mathcal{M}(a,b) \cap \{0,1\}^n| = (2^b + 1)^{2^a}.$$

Thus, the claim follows by Corollary 4.2.1. □

5 Sorting all inputs

In this section, we make use of Theorem 1 and Lemma 4.3 in order to construct a nearly logarithmic depth family of hypercubic sorting networks. At a high level, the construction is simply based on recursive merging: the input is partitioned into some number of equal-sized lists, each of these lists is sorted recursively, and the resulting 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. The following lemma proves that the merge step can itself be reduced to sorting.

Let δ^*, δ, and f be as defined in Section 3. In addition, let

$$\begin{aligned}\delta' &\stackrel{\text{def}}{=} \delta^* - \frac{2}{f(d)},\\ k_0 &\stackrel{\text{def}}{=} \lceil f(d)(\lg d+3)/\delta\rceil, \text{ and}\\ \sigma^* &\stackrel{\text{def}}{=} -2\lg\delta^* \approx 4.260.\end{aligned}$$

Lemma 5.1 For every positive integer d, and every integer k such that $k_0 \le k < d$, we have

$$M(\lfloor\delta' k\rfloor, d-\lfloor\delta' k\rfloor) \le S(k) + \Theta(d+(d-k)f(d)).$$

Proof: Let $n = 2^d$. By Theorem 1, there exists an n-input hypercubic comparator network $\mathcal{N}$ of depth $S(k)+\Theta(d+(d-k)f(d))$ such that

$$|Sort(\mathcal{N}) \cap \Pi(n)| \ge (1 - O(n^3 2^{-2^{\delta k}}))n!.$$

For d sufficiently large, this implies

$$\begin{aligned}|Sort(\mathcal{N})\cap\Pi(n)| &\ge (1-n^4 2^{-2^{\delta k}}))n!\\ &= (1-2^{2^{\lg d+2}-2^{\delta k}})n!\\ &\ge (1-2^{-2^{\delta k-1}})n!,\end{aligned}$$

where the last inequality holds because $\delta k \ge \delta k_0 \ge \lg d + 3$. The result now follows from Lemma 4.3 since

$$\begin{aligned}\left(2^{d-\lfloor\delta' k\rfloor}+1\right)^{2^{\lfloor\delta' k\rfloor}} &< (n^2)^{2^{\delta' k}}\\ &= 2^{2^{\delta' k+\lg d+1}},\end{aligned}$$

and

$$\begin{aligned}\delta k - 1 &= \delta' k + (\delta-\delta')k - 1\\ &= \delta' k + \frac{k}{f(d)} - 1\\ &> \delta' k + \lg d + 1.\end{aligned}$$

□

As a consequence of the preceding lemma, we can develop a recurrence for upper-bounding $S(d)$. For all a, $0 \le a \le d$, we have

$$S(d) \le S(d-a) + M(a, d-a).$$

Thus $S(d)$ is less than or equal to

$$\min_{0\le k<d} S(d-\lfloor\delta' k\rfloor) + M(\lfloor\delta' k\rfloor, d-\lfloor\delta' k\rfloor)$$

which is at most

$$\min_{k_0\le k<d} S(d-\lfloor\delta' k\rfloor) + S(k) + \Theta(d+(d-k)f(d)).$$

Let $S'(d)$ denote the best upper bound on $S(d)$ obtainable via this recurrence. If $0 \le d \le 1 + \lfloor 1/\delta'\rfloor$, we have $S'(d) = \Theta(1)$. For larger values of d, $S'(d)$ is less than or equal to

$$\min_{k_0\le k<d} S'(d-\lfloor\delta' k\rfloor) + S'(k) + \Theta(d+(d-k)f(d)).$$

In Appendix A it is proven that

$$S'(d) = \Theta(df(d)2^{\sqrt{\sigma^*\lg d}+\Theta(\sqrt{\lg d}/f(d))}\sqrt{\lg d}).$$

It is now straightforward to optimize the choice of the constant c in the definition of $f(d)$. In particular, setting $c = \frac{1}{2}$ provides our best upper bound on $S(d)$, namely,

$$S(d) = O(d2^{\sqrt{\sigma^*\lg d}}\lg d).$$

6 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 $1 - 2^{-n^\epsilon}$ for some constant $\epsilon > 0$.

One unfortunate characteristic of our hypercubic sorting network construction is its lack of uniformity. In particular, no polynomial-time algorithm is known for generating the family of networks for which existence has been established. There is reason to believe that such a generation algorithm might exist; providing a randomized polynomial-time generation algorithm is a straightforward extension of the results proven in the paper.

A Analysis of the recurrence

Throughout this section, let δ^*, δ', f, k_0, and σ^* be as defined in Sections 3 and 5. Also, let $\sigma' \overset{\text{def}}{=} -2\lg\delta'$. Note that $\sigma' = \sigma^*(1+\Theta(1/f(d))$. Consider the recurrence defined by $T(d) = 1$ if $0 \leq d \leq 1 + \lfloor 1/\delta' \rfloor$ and

$$T(d) = \min_{1 \leq k < d} T(d - \lfloor \delta' k \rfloor) + T(k) + df(d)$$

otherwise. In this section, we will prove the following theorem:

Theorem 2 The solution to the above recurrence is

$$T(d) = \Theta(df(d)2^{\sqrt{\sigma' \lg d}}\sqrt{\lg d}).$$

Corollary 2.1 The solution to the recurrence for $S'(d)$ in Section 5 is

$$\begin{aligned} S'(d) &= \Theta(df(d)2^{\sqrt{\sigma' \lg d}}\sqrt{\lg d}) \\ &= \Theta(df(d)2^{\sqrt{\sigma^* \lg d}+\Theta(\sqrt{\lg d}/f(d))}\sqrt{\lg d}). \end{aligned}$$

Proof: Consider the first equality. The only significant difference between the recurrence for $S'(d)$ and the recurrence for $T(d)$ is the lower bound on k; we require $k \geq k_0$ in the former case but only $k \geq 1$ for the latter recurrence. Thus any lower bound proven for $T(d)$ also applies to $S'(d)$. We now argue that the upper bound proven for $T(d)$ also applies to $S'(d)$. In order to obtain the upper bound on $T(d)$ (see the proof of Lemma A.1 below) we set $k = \lfloor d/2^{\sqrt{\sigma' \lg d}} \rfloor$ in the recurrence for $T(d)$. This choice of k is greater than or equal to k_0 for d sufficiently large. For $d = O(1)$, we can sort in constant depth.

For the second equality, note that

$$\begin{aligned} \sqrt{\sigma' \lg d} &= \sqrt{\sigma^* \lg d}(1+\Theta(1/f(d))) \\ &= \sqrt{\sigma^* \lg d} + \Theta(\sqrt{\lg d}/f(d)). \end{aligned}$$

□

The upper and lower bounds implied by Theorem 2 will be established separately. The upper bound is addressed by the following lemma.

Lemma A.1 There exists a positive constant c_0 such that

$$T(d) \leq c_0 df(d)2^{\sqrt{\sigma' \lg d}}\sqrt{\lg d} + 1$$

for all $d \geq 1$.

Proof: We prove the claim by induction on d. Note that the claim holds for $1 \leq d < d_0$, where d_0 denotes an arbitrary positive constant. We are free to choose d_0 sufficiently large that

$$\left\lfloor \frac{d}{2^{\sqrt{\sigma' \lg d}}} \right\rfloor \geq 1/\delta'$$

for all $d \geq d_0$. Now fix $d \geq d_0$ and assume that the claim of the lemma holds for all smaller values of d. Setting $k = k^* = \left\lfloor d/2^{\sqrt{\sigma' \lg d}} \right\rfloor$ in the recurrence for $T(d)$, and letting $d^* = d - \lfloor \delta' k^* \rfloor$, we obtain

$$T(d) \leq T(d^*) + T(k^*) + df(d). \tag{1}$$

Note that $1 \leq k^* < d$. Hence, the induction hypothesis implies that

$$\begin{aligned} T(d^*) &\leq c_0 d^* f(d^*)2^{\sqrt{\sigma' \lg d^*}}\sqrt{\lg d^*} + 1 \\ &\leq c_0 d^* f(d)2^{\sqrt{\sigma' \lg d}}\sqrt{\lg d} + 1 \\ &= c_0 df(d)\sqrt{\lg d}(2^{\sqrt{\sigma' \lg d}} - \delta') \\ &\quad + O(f(d)2^{\sqrt{\sigma' \lg d}}\sqrt{\lg d}). \end{aligned}$$

In the above inequality, we have also made use of the fact that f is monotonically nondecreasing. A second application of the induction hypothesis implies that

$$\begin{aligned} T(k^*) &\leq c_0 k^* f(k^*)2^{\sqrt{\sigma' \lg k^*}}\sqrt{\lg k^*} + 1 \\ &\leq c_0 \left(\frac{d}{2^{\sqrt{\sigma' \lg d}}}\right) f(d)2^{\sqrt{\sigma' \lg d - \sigma' \sqrt{\sigma' \lg d}}} \\ &\quad \cdot \sqrt{\lg d - \sqrt{\sigma' \lg d}} + 1. \end{aligned}$$

Note that

$$\left(\sqrt{\lg d} - \sqrt{\sigma'}/2\right)^2 = \lg d - \sqrt{\sigma' \lg d} + \sigma'/4.$$

and so $\sqrt{\lg d - \sqrt{\sigma' \lg d}} \leq |\sqrt{\lg d} - \sqrt{\sigma'}/2|$. For d sufficiently large (i.e., assuming that the constant

d_0 is chosen sufficiently large), we have $\sqrt{\lg d} > \sqrt{\sigma'}/2$ and so

$$\begin{aligned} T(k^*) &\leq c_0 df(d)2^{-\sigma'/2}(\sqrt{\lg d} - \sqrt{\sigma'}/2) + 1 \\ &= c_0\delta' df(d)(\sqrt{\lg d} - \sqrt{\sigma'}/2) + 1. \end{aligned}$$

Substituting our upper bounds for $T(d^*)$ and $T(k^*)$ into Equation (1), we obtain

$$\begin{aligned} T(d) \leq\ & c_0 df(d)2^{\sqrt{\sigma' \lg d}}\sqrt{\lg d} - c_0\delta' df(d)\sqrt{\sigma'}/2 \\ & + df(d) + O(f(d)2^{\sqrt{\sigma' \lg d}}\sqrt{\lg d}). \end{aligned}$$

For sufficiently large choices of the positive constants c_0 and d_0, the preceding inequality implies that

$$T(d) \leq c_0 df(d)2^{\sqrt{\sigma' \lg d}}\sqrt{\lg d} + 1,$$

as required. □

The lower bound of Theorem 2 is established by the following lemma.

Lemma A.2 There exists a positive constant c_0 such that

$$T(d) \geq c_0 df(d)2^{\sqrt{\sigma' \lg d}}\sqrt{\lg d}$$

for all $d \geq 1$.

Proof: The proof is by induction on d. Let d_0 denote an integer constant strictly greater than $1 + \lfloor 1/\delta' \rfloor$. The claim of the lemma is certainly satisfied for $d < d_0$. Now fix $d \geq d_0$ and assume that the claim holds for all smaller values of d. In the following, let

$$\begin{aligned} r(n) &= f(n)2^{\sqrt{\sigma' \lg n}}\sqrt{\lg n}, \text{ and} \\ s(n) &= nr(n), \end{aligned}$$

for all integers $n > 0$. Letting $d' = d - \lfloor \delta' k \rfloor$, $1 \leq k < d$, we begin by investigating the difference between $s(d)$ and $s(d')$. We have

$$\begin{aligned} d' &= d(1 - \lfloor \delta' k \rfloor / d), \\ \lg d' &= \lg d \left[1 - \Theta\left(\frac{k}{d \lg d}\right)\right], \\ \sqrt{\lg d'} &= \sqrt{\lg d} \left[1 - \Theta\left(\frac{k}{d \lg d}\right)\right], \\ 2^{\sqrt{\sigma' \lg d'}} &= 2^{\sqrt{\sigma' \lg d}} \left[1 - \Theta\left(\frac{k}{d\sqrt{\lg d}}\right)\right], \\ f(d') &= f(d) \left[1 - \Theta\left(\frac{k}{d\sqrt{\lg d}}\right)\right], \text{ and} \\ r(d') &= r(d) \left[1 - \Theta\left(\frac{k}{d\sqrt{\lg d}}\right)\right]. \end{aligned}$$

Thus,

$$s(d') = s(d) - \lfloor \delta' k \rfloor r(d) - \Theta(r(d)k/\sqrt{\lg d}).$$

and for d sufficiently large there exists a positive constant c_1 such that

$$s(d') \geq s(d) - \lfloor \delta' k \rfloor r(d) - c_1 r(d)k/\sqrt{\lg d}. \quad (2)$$

Letting $t(d,k) = \sqrt{\sigma' \lg d} + \lg k - \lg d$, we can develop a useful approximation for $s(k)$, $1 \leq k < d$, as follows. Note that $\sqrt{\sigma' \lg d} - \lg d \leq t(d,k) < \sqrt{\sigma' \lg d}$. Several of the Θ-bounds that follow contain functions that include a $t(d,k)$ factor. In such cases, the function may be asymptotically negative. If $g(n)$ is asymptotically negative then the expression $\Theta(g(n))$ should be interpreted as $-\Theta(|g(n)|)$. We have

$$\begin{aligned} \lg k &= (\lg d - \sqrt{\sigma' \lg d}) \\ &\quad \cdot(1 + \Theta(t(d,k)/\lg d)) \\ \sqrt{\lg k} &= (\sqrt{\lg d} - \sqrt{\sigma'}/2)(1 - \Theta(1/\lg d)) \\ &\quad \cdot(1 + \Theta(t(d,k)/\lg d)), \\ 2^{\sqrt{\sigma' \lg k}} &= \delta' 2^{\sqrt{\sigma' \lg d}}(1 - \Theta(1/\sqrt{\lg d})) \\ &\quad \cdot(1 + \Theta(t(d,k)/\sqrt{\lg d})), \\ f(k) &= f(d)(1 - \Theta(1/\sqrt{\lg d})) \\ &\quad \cdot(1 + \Theta(t(d,k)/\sqrt{\lg d})), \text{ and} \\ r(k) &= \delta' r(d)(1 - \Theta(1/\sqrt{\lg d})) \\ &\quad \cdot(1 + \Theta(t(d,k)/\sqrt{\lg d})). \end{aligned}$$

Thus,

$$\begin{aligned} s(k) &= \delta' kr(d)(1 - \Theta(1/\sqrt{\lg d})) \\ &\quad \cdot(1 + \Theta(t(d,k)/\sqrt{\lg d})) \\ &= \delta' kr(d) - \Theta(kr(d)/\sqrt{\lg d}) \\ &\quad + \Theta(kr(d)t(d,k)/\sqrt{\lg d}), \end{aligned}$$

and for d sufficiently large there exist positive constants c_2 and c_3 such that

$$\begin{aligned} s(k) \geq\ & \delta' kr(d) - c_2 kr(d)/\sqrt{\lg d} \\ & + c_3 kr(d)t(d,k)/\sqrt{\lg d}. \end{aligned} \quad (3)$$

Using the recurrence for $T(d)$ along with two applications of the induction hypothesis, we have

$$\begin{aligned} T(d) &= \min_{1\le k<d} T(d') + T(k) + df(d) \\ &\geq \min_{1\le k<d} c_0 s(d') + c_0 s(k) + df(d). \end{aligned}$$

We can now apply the bounds of Equation (2) and Equation (3) to obtain

$$\begin{aligned} T(d) &\geq \min_{1\le k\le d} c_0 s(d) + df(d) \\ &\quad + c_0(c_3 t(d,k) - c_1 - c_2)kr(d)/\sqrt{\lg d} \\ &= \min_{1\le k\le d} c_0 s(d) + df(d) \\ &\quad + c_0(c_3 t(d,k) - c_1 - c_2)2^{t(d,k)} df(d). \end{aligned}$$

If $t(d,k) \geq c_4 \overset{\text{def}}{=} (c_1+c_2)/c_3$ then it is immediate that $T(d) \geq c_0 s(d)$, and the induction is complete. On the other hand, if $2^{t(d,k)} < c_5 \overset{\text{def}}{=} 2^{c_4}$ then

$$c_0(c_3 t(d,k) - c_1 - c_2)2^{t(d,k)} > -c_0(c_1+c_2)c_5.$$

Setting $c_0 \leq [(c_1+c_2)c_5]^{-1}$ we once again have $T(d) \geq c_0 s(d)$, as required. □

References

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

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

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

[4] V. Chvátal. Personal communication.

[5] R. E. Cypher. Theoretical aspects of VLSI pin limitations. Technical Report RJ7115, IBM Almaden Research Center, November 1989.

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

[7] 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.

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

[9] C. G. Plaxton and T. Suel. A lower bound for sorting networks based on the shuffle permutation. Technical Report TR–92–07, University of Texas at Austin, Department of Computer Science, March 1992.