Constant Time Parallel Sorting: An Empirical View

William Gasarch[*]
Univ. of MD, College Park

Evan Golub[†]
Univ. of MD, College Park

Clyde Kruskal[‡]
Univ. of MD, College Park

1 Introduction

It is well known that sorting can be done with $O(n \log n)$ comparisons. It is also known that (in the comparison decision tree model) sorting requires $\Omega(n \log n)$ comparisons.

What happens if you allow massive parallelism? In the extreme case you can sort n elements in one round by using $\binom{n}{2}$ processors to make all the comparisons at once. It is easy to show that sorting in one round *requires* $\binom{n}{2}$ processors. Can you sort in two rounds with a subquadratic number of processors? What about k rounds? We survey the known literature and discuss simulations of these algorithms that we have carried out. One of our main points will be that *nonconstructive algorithms can be useful.*

We use the parallel decision tree model introduced by Valiant [25]. If p processors are used then every node is a set of p comparisons and has 2^p children corresponding to all possible answers. We think of a node as having information about how the comparisons that led to that node were answered (formally a DAG on $\{x_1, \ldots, x_n\}$) and all information derivable from that information (formally the transitive closure of that dag). The

[*]Dept. of C.S. and Inst. for Adv. Comp. Stud., University of MD., College Park, MD 20742, U.S.A. Supported in part by NSF grant CCR–9732692. (Email: gasarch@cs.umd.edu.)

[†]Dept. of C.S., University of MD., College Park, MD 20742, U.S.A. (Email: egolub@acm.org.)

[‡]Dept. of C.S., University of MD., College Park, MD 20742, U.S.A. Supported in part by NSF grant CCR–9732692. (Email: kruskal@cs.umd.edu.)

model does not count the cost of communication between processors, nor does it count the cost of transitive closure. The model does capture how hard it is to gather the *information* needed to sort. Also, lower bounds in our model will apply to models that take these factors into account. For more realistic models of parallelism see any current textbook on parallel algorithms, e.g. [1, 16].

The first round of a p-processor algorithm takes $x_1, \ldots, x_n$ about which nothing is known and makes p comparisons. This can be represented as an undirected graph G on n vertices with p edges. Hence the search for parallel sorting algorithms will involve finding graphs G that have nice properties. Most of our algorithms depend on versions of the following two lemmas, which we state informally:

1. Some undirected graph G with property P exists and does not have too many edges.

2. Let $G = (V, E)$ be a graph with $V = \{x_1, \ldots, x_n\}$ that has property P. Let G' be *any* acyclic orientation of G and let H be the transitive closure of G'. The graph H does not have too many edges.

The literature has many algorithms to sort in constant time. Some are nonconstructive. We have undertaken an empirical study of these algorithms by simulating most of them. This paper will present the results of that study. We include proof sketches to indicate the algorithms used. A companion paper [12] discusses the proofs in more detail.

2 Definitions and Notation

There are several types of sorting algorithms.

Def 2.1

1. A *nonconstructive algorithm for sorting n elements in k rounds* is an algorithm that is proven to exist, but its existence proof does not reveal how to produce it. For example, the graph on n vertices that represents the first round may be proven to exist by the probabilistic method [5, 23].

2. A *constructive algorithm for sorting in k rounds* is a sequence of algorithms $\mathcal{A}_n$ with the following properties: (1) The algorithm $\mathcal{A}_n$ sorts n elements in k rounds. (2) There is a polynomial time algorithm that, given n (in unary), produces $\mathcal{A}_n$.

3. A *randomized algorithm for sorting in k rounds* is a sequence of randomized algorithms $\mathcal{A}_n$ with the following properties: (1) The algorithm $\mathcal{A}_n$ sorts n elements in k rounds. (2) There is a polynomial time algorithm that, given n (in unary), produces $\mathcal{A}_n$. Each time you run

the algorithm the number of processors may vary since it is randomized. We will be concerned with the expected number of processors. One could instead fix the number of processors and be concerned with the number of rounds. This is studied in [16]. Note that we are dealing with *constructive randomized algorithms*. We do not know of any *nonconstructive randomized algorithms*.

We noted above that the model assumes transitive closure is free. Some of our algorithms work with the weaker assumption that only a partial transitive closure is free.

Def 2.2

1. Given a directed graph G the *2-step transitive closure* is the graph formed as follows: If in our original comparison graph we have (x, y) and (y, z) then we will add to that graph (x, z). Note that if we have (x, y), (y, z), and (z, w) we *do* add (x, z) and we *do* add (y, w) but we *do not* add (x, w).

2. Let $d \geq 2$. Given a directed graph G the *d-step transitive closure* is the graph defined inductively as follows: (1) the 2-step transitive closure is as above. (2) the d-step transitive closure is the 2-step transitive closure of the $(d-1)$-step transitive closure.

Def 2.3

1. $\text{sort}(k, n)$ is the number of processors needed to sort n elements in k steps. The algorithm may be nonconstructive.

2. $\text{csort}(k, n)$ is the number of processors needed to sort n elements in k steps by means of a constructive algorithm.

3. $\text{sort}(k, n, d)$ is the number of processors needed to sort n elements in k steps by means of an algorithm that only uses d-step transitive closure. The algorithm may be nonconstructive.

4. $\text{csort}(k, n, d)$ is the number of processors needed to sort n elements in k steps by means of a constructive algorithm that only uses d-step transitive closure.

5. $\text{rsort}(k, n)$ is the expected number of processors needed to sort n elements in k steps by means of a randomized algorithm.

Note 2.4 When we use order notation we take k to be a constant. Hence a statement like "$\text{sort}(k, n) = O(n^{1+1/k}(\log n)^{2-2/k})$" means that the multiplicative constant might depend on k.

We survey all known upper bounds on the quantities in Definition 2.3. Our goal is that the reader (1) learns that there are many interesting constant-time parallel sorting algorithms in the literature, and (2) learns what happens when these algorithms are simulated.

3 Empirical Methodology

Since multiprocessor machines with enough processors to run these algorithms do not currently exist, we performed our empirical studies using uniprocessors machines. This was accomplished by implementing all algorithms using a for-loop to represent a single round. Within the loop, each iteration represented a unique processor. Care was taken to assure that later iterations had no access to information obtained during the earlier ones.

One limitation that this methodology presented was run-time. There were two factors involved within this. First, as the number of elements to be sorted increases, the run-time increases since we are using a single processor. Second, although the model assumes that transitive closure is free, the run-time cost of this on a uniprocessor system is high.

As a result, experiments were performed with values of n ranging from as small as 2^6 to no more than 2^{14}. However, it should be noted that we were able to make many observations within this range.

Another issue that needed to be dealt with was the generation of inputs as well as the generation of graphs in some cases. To offset the known problems with random number generation, in all experiments, multiple inputs and multiple graphs were used. The best, worst, and average of the results on these inputs and graphs were all observed during the analysis. When looking for results, we focused on the worst-cases of the trials. We ran each algorithm (and in the case of algorithms that used randomly generated graphs, on each graph) on between 10 and 100 different inputs as time allowed.

4 Nonconstructive Methods

4.1 The First Nonconstructive Algorithm

The first k-round sorting algorithm that uses a subquadratic number of processors is due to Haggkvist and Hell [14]. They showed that $\mathrm{sort}(k,n) \le O(n^{\alpha_k} \log_2 n)$ where $\alpha_k = \frac{3 \cdot 2^{k-1}-1}{2^k-1}$. In particular this implies $\mathrm{sort}(2,n) \le O(n^{5/3} \log n)$. Bollobás and Thomason [9] improved the $k=2$ case by showing $\mathrm{sort}(2,n) \le O(n^{3/2} \log n)$.

Theorem 4.1 ([9, 14])

1. $\mathrm{sort}(2,n) \le O(n^{5/3} \log n)$ *[14].*
2. $\mathrm{sort}(k,n) \le O(n^{\alpha_k} \log n)$ *where* $\alpha_k = \frac{3 \cdot 2^{k-1}-1}{2^k-1}$ *[14].*
3. $\mathrm{sort}(2,n) \le O(n^{3/2} \log n)$ *[9].*

Algorithm sketch: We sketch the first result. The second result uses induction with the first result as its base case. The third result is similar to the first, but non-trivial. Let $\alpha = \frac{5}{3}$, $p = \lfloor n^{2-\alpha} \rfloor$, $q = \lfloor n^{4-2\alpha} \rfloor$, and

$r = \lfloor 2n^{4\alpha-6} \log_2 n \rfloor$. Let $A = |\{G : G$ has n vertices and pqr edges $\}|$. Let G be a graph from A. Consider the following algorithm.

1. (Round 1) Compare $x_i : x_j$ iff (i,j) is an edge of G. (This takes $pqr = O(n^\alpha \log n)$ comparisons.) Let G' be the orientation of G obtained by directing i to j iff $x_i < x_j$. Let H be the transitive closure of G'.
2. (Round 2) Compare all $x_i : x_j$ such that (i,j) is not an edge of H.

One can show that there exists graphs $G \in A$ such that round 2 takes $O(n^\alpha \log n)$ comparisons. In fact, one can show that *most* graphs in A have this property.
▮

Nota Bene 4.2 Some authors have credited [7] or [9] with the result $\text{sort}(k,n) = O(n^{1+1/k} \log n)$. This citation is incorrect and this result is not even known to be true. However, Bollobás [6] later obtained $\text{sort}(k,n) = O(n^{1+1/k} \frac{(\log n)^{2-2/k}}{(\log\log n)^{1-1/k}})$ (see Section 4.3).

Empirical Note 4.3 We did not code this algorithm up. The algorithms are nonconstructive in that the graphs needed to represent the rounds are proven to exist but no method is provided to construct them. This is *not* an obstacle (see Section 4.2). In these algorithms a graph is picked at random from a set of graphs that have a certain number of edges. This is hard to program. As we will see, other types of probabilistic methods are easy to code.

4.2 Expander Graphs

Pippenger [20] showed that $\text{sort}(k,n) = O(n^{1+1/k}(\log n)^{2-2/k})$.

Def 4.4 [20] Let $1 \le a \le n/2$. An a-expanding graph is a graph in which for any two disjoint sets of vertices of size $a+1$, there is at least one edge between the two sets.

Lemma 4.5 ([20]) *For $1 \le a \le n/2$ there exists an a-expanding graph with $O(\frac{n^2 \log n}{a})$ edges.*

Algorithm sketch: Assume you have a coin that has probability $p = \frac{2\ln n}{a}$ of being heads. Create a graph on n vertices as follows: for each $\{i,j\}$, flip the coin. Put the edge $\{i,j\}$ into the graph iff the coin is heads. The probability that the graph will be an a-expander graph with $O(\frac{n^2 \log n}{a})$ edges is nonzero (actually close to 1). Hence such a graph exists. ▮

Lemma 4.6 ([20]) *If n elements are compared according to the edges of an a-expander graph, then there will be at most $O(a \log n)$ candidates remaining for any given rank.*

From Lemma 4.6 it is easy to prove the following:

Lemma 4.7 ([20]) *If n elements are compared according to the edges of an a-expander graph, then they can be partitioned into $O(\frac{n}{a \log n})$ sets, each containing $O(a \log n)$ elements, such that the relationship between any pair of elements is known unless they both belong to a common set.*

Theorem 4.8 ([20]) $\mathrm{sort}(k, n) = O(n^{1+1/k}(\log n)^{2-2/k})$.

Algorithm sketch: We prove this by induction on k. For $k = 1$ this is trivial. Assume the theorem for $k - 1$.

Let G be an a-expanding graph that is shown to exist by Lemma 4.5. We will pick the value of a later.

1. (Round 1) Compare $x_i : x_j$ iff (i, j) is an edge of G. (This takes $O(\frac{n^2 \log n}{a})$ comparisons.) Let G' be the orientation of G obtained by directing i to j iff $x_i < x_j$. Let H be the transitive closure of G'.

2. (Rounds $2, \ldots, k$) Using Lemma 4.7 one can show that $\{x_1, \ldots, x_n\}$ can be partitioned into $O(\frac{n}{a \log n})$ groups of size $O(a \log n)$ such that all comparisons between different groups are known. Sort the groups inductively in $k - 1$ rounds. This takes

$$O(\frac{n}{a \log n}(a \log n)^{1+\frac{1}{k-1}}(\log(a \log n))^{2-\frac{2}{k-1}})$$

processors.

To achieve the result set $a = \Theta(\frac{n^{1-1/k}}{(\log n)^{1-2/k}})$. ∎

Empirical Note 4.9 When implementing the algorithm, the probability p which was to be used to generate the a-expanding graphs is based on the value for a which is in turn based on n. As a starting point, we took p to be the exact value specified in the paper, $p = \frac{2 \ln n}{a}$. After gathering results on 2-round sorting for values of n ranging between 100 and 5000 using that value for p, we charted the number of processors that were used in each of the rounds in the worst case and found that more work was being done in the first round than in the second. This implied that smaller graphs might be better. From this, we generated experimental results based on p multiplied by a constant factor between 0.01 and 2.00. We refer to this constant factor as C_p. These results led to the observation that a good value for C_p would be somewhere between 0.2 and 0.4. Further experiments lead to 0.36 as the best value for C_p. Figure 1 shows results with C_p values between 0.01 and 0.60.

Figure 1

Figure 2 Figure 3

We can see in Figures 2 and 3 that the ratio between the number of processors predicted by Pippenger and the number of processors used by the simulation when using C_p=0.36 levels off quickly, which agrees with Pippenger's asymptotic analysis.

4.3 Super Expander Graphs

Alon and Azar [3] showed that $\text{sort}(2, n) = O(n^{3/2} \frac{\log n}{\sqrt{\log \log n}})$. Bollobás [6] extended this to show that $\text{sort}(k, n) = O(n^{1+1/k} \frac{(\log n)^{2-2/k}}{(\log \log n)^{1-1/k}})$

All these results use graphs similar to the a-expander graphs discussed in Section 4.2. We sketch the algorithm of Alon and Azar and then make some brief comments about Bollobás 's algorithm.

We define a subset of a-expander graphs that has additional expanding properties. The following definition is implicit in [3].

Def 4.10 Let $a, n \in \mathsf{N}$ and $a = \Omega(\log n)$. A graph G on n vertices is an *a-super-expander* if the following hold.

1. If A and B are disjoint subsets of vertices with a vertices each then some $v \in B$ has at least $\log_2 n$ neighbors in A.
2. Let $x \le a/e^{\sqrt{\log_2 n}}$. If A and B are disjoint sets such that $|A| = x$ and $|B| = x(\log_2 n)^{1/4}$, then each $v \in A$ has at least $\log_2 n$ neighbors in B.

The following lemma asserts that there exists small a-super-expander graphs. It is similar to Lemma 4.5; however we will be using it with a different value of a to obtain a better upper bound on $\mathrm{sort}(k, n)$.

Lemma 4.11 ([3]) *There exists an a-super-expanding graph with $O(\frac{n^2 \log n}{a})$ edges.*

Algorithm sketch: Assume you have a coin that has probability $p = \Theta(\frac{\log n}{a})$ of being heads. Create a graph on n vertices as follows: for each $\{i, j\}$, flip the coin. Put the edge $\{i, j\}$ into the graph iff the coin is heads. The probability that the graph will be an a-super-expander with $O(\frac{n^2 \log n}{a})$ edges is nonzero (actually close to 1). Hence such a graph exists. ∎

Lemma 4.12 ([3]) *If n elements are compared according to the edges of an a-super-expanding graph, then there will be at most $O(a \log n / \log\log n)$ candidates remaining for any given rank.*

Theorem 4.13 ([3]) $\mathrm{sort}(2, n) = O(n^{3/2} \frac{\log n}{\sqrt{\log\log n}})$

Algorithm sketch: This is similar to the $k = 2$ case of Theorem 4.8. The value of a needed is $a = \Theta(\sqrt{n \log\log n})$. ∎

Theorem 4.14 ([6]) $\mathrm{sort}(k, n) = O(n^{1+1/k} \frac{(\log n)^{2-2/k}}{(\log\log n)^{1-1/k}})$

Algorithm sketch: A rather complicated type of graph is defined which will, if used to guide comparisons, yield much information. Let

$$p = \Theta(\frac{n^{1/k}(\log n)^{2-2/k}}{n(\log\log n)^{1-1/k}}).$$

Assume you have a coin that has probability p of being heads. A graph on n vertices as follows: for each $\{i, j\}$, flip the coin. Put the edge $\{i, j\}$ into the graph iff the coin is heads. The probability that the graph will be of this type and have $O(n^{1+1/k} \frac{(\log n)^{2-2/k}}{(\log\log n)^{1-1/k}})$ edges is nonzero (actually close to 1). We use this type of graph in round 1 and then proceed inductively. ∎

Empirical Note 4.15 We did not code this algorithm up. For the values of n that we are looking at it is not clear that the $(\log\log n)^{1-1/k}$ factor savings would be visible. Also, the algorithm to generate these graphs is the same as that for the algorithms in section 4.2— generate a graph by putting edges in with probability $p = \Theta(\frac{\log n}{a})$. In our coding up of Pippenger's algorithm we had to fine-tune the value of p. We would do the same thing here. It is likely we would obtain very similar results. In fact, because of the similarities in the methods, it is quite possible that we *did* code up this algorithm while coding up Pippenger's.

5 Constructive Methods

5.1 Merging and Sort

Let merge(k, n) be the number of processors needed to merge two lists of n elements in k rounds.

Haggkvist and Hell [15] present constructive proofs for the following upper bounds: merge$(k, n) = \Theta(n^{1+1/(2^k-1)})$ and csort$(k, n) = O(n^{1+\sqrt{2}/k})$. Their sorting algorithm uses parallel merging. The paper gives matching upper and lower bounds for merging. While all that was needed was an upper bound for merging, knowing the exact bound allows us to know that the sorting algorithm cannot be improved via an improvement to the bound on merging.

Lemma 5.1 ([15]) merge$(k, n) = O(n^{\frac{2^k}{2^k-1}})$

Algorithm sketch:

The algorithm given for merging two ordered lists of n elements is to partition each list into groups, and then do a pairwise comparison of the first element of each group in the first list with the first element of each group in the second list. After doing these comparisons, there will be a small number of groups whose members are still unordered relative to one another. To prove this they consider the following graph: V is the set of groups, and an edge is placed between A and B if there is an $x \in A$ and a $y \in B$ such that the ordering $x : y$ is not known. They show that this graph is planar and thus linear in size.

Haggkvist and Hell establish that a group size of $O(n^{1/3})$ is optimal for two round parallel merging, giving merge$(n, 2) = O(n^{4/3})$. By applying induction on the merging of the groups whose orientation was not previously determined by the comparison of the first elements of each group, they derive the generalization merge$(k, n) = \Theta(n^{\frac{2^k}{2^k-1}})$ ▮

Note 5.2 Haggkvist and Hell also showed that merge$(k, n) = \Omega(n^{\frac{2^k}{2^k-1}})$.

Theorem 5.3 ([15])

1. $\text{csort}(3, n) = O(n^{8/5})$.
2. $\text{csort}(4, n) = O(n^{20/13})$.
3. $\text{csort}(5, n) = O(n^{28/19})$.
4. $\text{csort}(k, n) = O(n^{1+\sqrt{2}/k})$.

Algorithm sketch: The algorithm to sort a list of values in k rounds is based on using some number of rounds j to partition the list and sort each partition, and then use the remaining $k - j$ rounds to do a pairwise merge of those partitions. In the 3 round case, the list is partitioned into groups of size $O(n^{2/5})$ and each partition is then sorted in one round using $O(n^{6/5})$ processors per partition, or a total of $O(n^{8/5})$ processors. Then in the two remaining rounds, a pairwise merging of the $O(n^{2/5})$ groups would produce all information required to fully order the original n values.

The other results are similar. In each case the calculation of the optimal value of j is nontrivial. Let s_k denote the smallest value such that a j exists that allows one to sort n numbers in k-rounds with $O(n^{s_k})$ processors. The following recurrence allows one to find s_k for any particular k; however, it has no closed form.

$$s_{k+1} = \min\{\frac{2(2^j - 1)s_{k+1-j} - 2^j}{(2^j - 1)s_{k+1-j} - 1} : j > 0 \wedge s_{k+1-j} \geq \frac{2^j}{2^j - 1}\}$$

From this one can derive the approximation $\text{csort}(k, n) = O(n^{1+\sqrt{2}/k})$. The calculation is not straightforward or tight. ∎

Note 5.4 Assume that we knew $\text{csort}(2, n) \leq n^{\alpha} \log n$. Then the recurrence in Theorem 5.3 could be modified to get results of the form $\text{csort}(k, n) = O(n^{\alpha_k} \log n)$. The results obtained would be better than those of Theorem 5.3 for small k. It is not clear what would happen asymptotically.

Empirical Note 5.5 The algorithm specifies the size of the groups into which the values should be partitioned. After this partitioning, two rounds are used to accomplish a pairwise merge across these partitions. The final round is used to answer the remaining questions. We began by partitioning into groups of exactly the specified size. The results when tested on values of n up to 2000 showed that there was a difference in the number of processors used in each of the two rounds used for merging.

To see if that difference would disappear as the value of n increased, we extended testing to values of n up to 10,000. The results showed that the difference remained. This could have indicated that (as with Pippenger)

Figure 4

Figure 5

Figure 6

Figure 7

Figure 8

some improvement might have been attainable by bringing the number of processors used in the two merging rounds closer together.

By varying the size of the partitions as well as the size of the groups during merging by a constant multiplicative factor, we found that small changes did not significantly affect the end result and that large changes had detrimental effects. Additionally, the points at which the two rounds used the same number of processors differed. Additionally, the number of processors required in the final round could not be predictably balanced with either of these two rounds. Figures 6, 7, and 8 demonstrate these differences for several values of n.

Although it was possible in some cases to bring the number of processors used in these rounds closer together, there did not appear to be a predictable way in which to optimize this. Additionally, the results shown in Figures 4 and 5 had the same growth rate as the formula predicts.

Figure 9 graphs the ratio of a formula's predictions with the empirical results as n increases. The ratio is relatively constant.

When moving to the case of sorting in 4 rounds, we discovered some interesting behavior. Specifically, for values of n less than 8192, the simulation was behaving much worse than predicted, but starting at 8192 it behaved as expected. See Figure 10.

Upon further investigation, we saw that as more rounds were introduced, for small values of n, groups of size 1 or 0 might be created for the last round, so the processors would not be fully utilized (in the last round). This

Figure 9

Figure 10

"saving" in the final round was made up for by using more than expected processors in the earlier rounds.

An additional issue which needs to be dealt with when implementing this algorithm is that it assumes that all groups will be of an equal size. There are several ways in which the groups can be padded to work with this limitation. In our simulations, we chose values for n that would partition evenly.

5.2 Attempts at the $k = 2$ case

Theorem 5.3 did not produce a constructive 2-round subquadratic sorting algorithm. This was eventually solved by Pippenger (see Section 5.3); however, before it was solved there were some interesting results that broke the $\binom{n}{2}$ barrier.

1. Haggkvist and Hell [14] showed $\mathrm{csort}(2, n) \le \frac{13}{15}\binom{n}{2}$. Their proof used the Peterson graph and balanced incomplete block designs.

2. Bollobás and Rosenfeld [8] showed $\mathrm{csort}(2, n) \le \frac{4}{5}\binom{n}{2}$. Their proof used the Erdos-Renyi graph [11] based on projective geometry.

Empirical Note 5.6 We did not code up these algorithms since there were far better nonconstructive ones available. Also, for the range of n we are discussing, it would be hard to tell $\binom{n}{2}$ from $\frac{4}{5}\binom{n}{2}$.

5.3 The First Constructive Subquadratic Algorithm for $k = 2$

In 1984 the first constructive 2-round subquadratic sorting algorithm was discovered by Pippenger [21] who showed $\mathrm{sort}(2, n) = O(n^{1.95})$. He never wrote it up; however, several references to it exist including one in [7]. A year later he improved this result and generalized to k rounds by developing the framework of expander graphs for sorting (see Section 4.2) and showing that the graphs constructed by Lubotzky, Phillips, and Sarnak [17] were a-expander graphs. These can be used to obtain sorting algorithms that are constructive, though not as good as the nonconstructive ones in Theorem 4.8.

Lemma 8 of [20] proves two things. We separate them out into two separate lemmas.

Lemma 5.7 ([20]) *Let G be a graph on n vertices. Let λ_i be the ith largest eigenvalue of the adjacency matrix. If $\lambda_1 = p + 1$ and $(\forall i \ge 2)[\lambda_i \le 2\sqrt{p}]$ then G is an $O(\frac{n}{\sqrt{p}})$-expanding graph.*

Lemma 5.8 ([17, 20]) *Let p, q be primes that are congruent to 1 mod 4. Assume $p < q$. There exists an explicitly constructed $O(\frac{q}{\sqrt{p}})$-expanding graph with $q + 1$ vertices and $O(pq)$ edges.*

Algorithm sketch: Lubotzky, Phillips, and Sarnak [17] constructed a $p+1$-regular graph G on $q+1$ vertices with the following properties: (1) the largest eigenvalue of the adjacency matrix, $p+1$, has multiplicity 1, and (2) all other eigenvalues have magnitude at most $2\sqrt{p}$. Clearly this graph is on $q+1$ vertices and has $O(pq)$ edges. By Lemma 5.7 this graph is an $O(\frac{q}{\sqrt{p}})$-expanding graph. ▮

Note 5.9 The graphs constructed by Lubotzky, Phillips, and Sarnak are somewhat complicated. They use graphs associated to certain groups.

Lemma 5.10 *Let $1 \le a \le n$. Let $\frac{n}{a}$ be sufficiently large. There is an explicitly constructed a-expanding graph G on n vertices of size $O(n^3/a^2)$.*

Algorithm sketch: We need to find primes p, q such that $(\frac{n}{a})^2 \le p \le 2(\frac{n}{a})^2$, $n \le q \le 2n$, and both p, q are congruent to 1 mod 4. Such exists for $\frac{n}{a}$ large by the Prime Number Theorem for arithmetic progressions (see [10] for example). Apply Lemma 5.8 to obtain a graph on $\Theta(n)$ vertices that is a $O(\frac{q}{\sqrt{p}})$-expanding, hence $O(\frac{n^3}{a^2})$-expanding. ▮

Theorem 5.11 $\text{csort}(k, n) \le O(n^{1+\frac{2}{(k+1)}}(\log n)^{2-\frac{4}{(k+1)}})$.

Algorithm sketch: This proof is similar to that of Theorem 4.8 except that we use Lemma 5.10 with

$$a = \Theta(\frac{n^{1-1/(2^k-1)}}{(\log n)^{2/(2^k-1)}}).$$

▮

Empirical Note 5.12 We did not code this algorithm up as it was somewhat complicated.

5.4 Two Simple Constructive Algorithms

Alon [2] showed that $\text{csort}(2, n, 2) = O(n^{7/4})$. His algorithm is simpler than that of Theorem 5.11. Since Alon's result is about limited closure sorting we will discuss it in Section 8; however by combining it with the recurrence in Theorem 5.3 he obtained improvements over Theorem 5.3. We give the first few improvements. More numbers can be generated; however, the asymptotic values do not improve.

Theorem 5.13 ([2])

1. $\text{csort}(2, n) = O(n^{7/4})$.
2. $\text{csort}(3, n) = O(n^{8/5})$.

3. $\mathrm{csort}(4,n) = O(n^{26/17})$.
4. $\mathrm{csort}(5,n) = O(n^{22/15})$.

Pippenger [20] noticed that a variant of Alon's algorithm actually yields $\mathrm{csort}(2,n) \le O(n^{5/3}\log n)$. (We will discuss this algorithm when discussing Alon's algorithm.) Golub [13] noticed that this could be combined with an easy modification of the recurrence in Theorem 5.3 (as described in the note following Theorem 5.3) to obtain a *simple* constructive algorithm which is better than that of Theorem 5.13. We give the first few improvements. More numbers can be generated; however, the asymptotic values do not improve.

Theorem 5.14

1. $\mathrm{csort}(2,n) = O(n^{5/3}\log n)$ *(use Pippenger's modification of Alon).*
2. $\mathrm{csort}(3,n) = O(n^{8/5})$ *(use Theorem 5.3).*
3. $\mathrm{csort}(4,n) = O(n^{3/2}\log n)$.
4. $\mathrm{csort}(5,n) = O(n^{23/16}\log n)$.

5.5 A Constructive Algorithm via Pseudo-Random Generators

Wigderson and Zuckerman [26] present a constructive proof that $\mathrm{sort}(k,n) = O(n^{1+1/k+o(1)})$. Their algorithm is based upon Pippenger's non-constructive sorting algorithm (See Section 4.2). Recall that Pippenger showed that small a-expander graphs were useful for sorting, and then showed that small a-expander graphs exist. The value of a taken for k round sorting was $a = \frac{n^{1-\frac{1}{k}}}{(\ln n)^{1-\frac{2}{k}}}$. Wigderson and Zuckerman present a constructive proof of the existence of a a-expander graphs with slightly worse values of a. They use the machinery of extractors and pseudo-random generators. Later authors improved this machinery and hence the results. The main results about sorting are summarized below.

Theorem 5.15 ([26]) $\mathrm{csort}(k,n) \le O(n^{1+1/k+o(1)})$.

There have been improvements in extractor technology which have lead to a better understanding of the $o(1)$ term in [22, 19].

Empirical Note 5.16 In this algorithm, there are a variety of interdependent formulas. In order for them to work, there are certain values which must be positive. By substituting in values to assure this, we find that 2^{16} is the smallest value of n for which the formulas can possibly work. Additionally, by writing a program to iterate through combinations of the variables used and the requirements upon these variables, we found that the first value that would work would be 2^{33}.

Empiricism played two roles in our study of this algorithm. Since previous algorithms had worked well on relatively small values of n we began coding our simulations and upon observing poor and inaccurate results, were motivated to investigate the mathematics in more detail. Empirical tests were then applied to the mathematical aspects of the algorithm in order to obtain more information about "sufficiently large values of n."

6 A Randomized Algorithm

Alon, Azar, and Vishkin showed $\text{rsort}(k, n) = O(n^{1+1/k})$. Their algorithm is fairly simple; however, the analysis requires care.

Theorem 6.1 ([4]) $\text{rsort}(k, n) = O(n^{1+1/k})$.

Algorithm sketch:

In the first round, $n^{1/k} - 1$ values are chosen at random and each is compared to all $n - 1$ other values. Between rounds, the n values are partitioned into $O(n^{1/k})$ blocks $(A_1, \ldots, A_{n^{1/k}})$ based on the now ordered list of $O(n^{1/k})$ values such that if $i < j$ then all members of A_i are less than members of A_j.

In the remaining $k-1$ rounds, each A_i is sorted. A careful analysis shows that the expected number of processors required to do this is $O(n^{1+1/k})$. ∎

Empirical Note 6.2 A question that arose during experiments with this algorithm was that of how the expected number of processors would translate into an actual number of rounds. Since an underestimate of the number of processors required would lead to additional rounds being needed it would be in our best interest to observe the behavior of the algorithm to assist us in the selection of a number of processors to use.

In the two round case, the average number of processors used in rounds 1 and 2, as well as the maximum number of processors used in round 2, are so close that if graphed, they all overlap with the line representing the formula. Similarly, many of the results in the three and four round cases are so close that they would overlap on a graph and appear as a single line or as a cluster of lines. It would be difficult to draw the graph as to show each line distinctly in gray scale.

The first round is deterministic and uses slightly less than $n^{1+1/k}$ processors. We call this the *formula value.* Not counting low order terms, it is a lower bound on how many processors the algorithm needs.

Figures 11, 12, and 13 show the maximum over all rounds, maximum of the averages of all rounds, the formula, and twice the formula.

There are still overlapping lines in the two round case, but the basic idea comes across: these empirical results give us reason to believe that if we allocate twice the expected number of processors we have a good chance of being able to avoid the requirement that additional rounds be used in the sort.

Figure 11

Figure 12

Figure 13

Empirical Note 6.3 There are still overlapping lines in the two round case, but the basic idea comes across: these empirical results give us reason to believe that if we allocate twice the expected number of processors we have a good chance of being able to avoid the requirement that additional rounds be used in the sort.

7 A Nonconstructive Algorithm for $\mathrm{sort}(2, n, d)$

Bollobás and Thomason [9] were the first ones to look at sorting with limited transitive closure. They used nonconstructive means, similar (though more complicated) to those we have seen in Theorems 4.1, 4.8, 4.13 and 4.14. Hence we omit even a sketch here.

Theorem 7.1 ([9]) $\mathrm{sort}(2, n, d) \le O(\frac{1}{2d} n^{1+\frac{d}{2d-1}} (\log n)^{1/2d-1})$.

Empirical Note 7.2 In these algorithms a graph is picked at random from a set of graphs that have a certain number of edges. This is very hard to program, so we did not do so.

8 A Constructive Algorithm for $\mathrm{sort}(2, n, 2)$

Alon [2] showed that $\mathrm{csort}(2, n, 2) = O(n^{7/4})$. He used techniques in projective geometry over finite fields to construct graphs. He used the eigenvalue methods of [24] to prove his graphs had the relevant properties. Pippenger used a variation of Alon's graphs to obtain $\mathrm{csort}(2,n) = O(n^{5/3}\log n)$.

Notation 8.1 If q is a prime power than F_q is the finite field on q elements.

Def 8.2 We refer to 1-step transitive closure as *Direct Implication.*

Def 8.3 [2] Let $d, q \in \mathsf{N}$ and let q be a prime power. The number d will be referred to as the *dimension.* The *Geometric Expander over F_q of dimension d* is the bipartite graph that we construct below:

1. Create a set of $d+1$ tuples of the following form

$$\begin{array}{ll}
(1, a_1, a_2, \ldots, a_{d+1}) & a_1, \ldots, a_{d+1} \in \{0, 1, \ldots, q-1\} \\
(0, 1, a_2, \ldots, a_{d+1}) & a_2, \ldots, a_{d+1} \in \{0, 1, \ldots, q-1\} \\
(0, 0, 1, a_3, \ldots, a_{d+1}) & a_3, \ldots, a_{d+1} \in \{0, 1, \ldots, q-1\} \\
\vdots & \\
(0, 0, 0, \ldots, 0, 1, a_{d+1}) & a_{d+1} \in \{0, 1, \ldots, q-1\}
\end{array}$$

   Note that each tuple represents a hyperplane in $d+1$ space over F_q. (Alternatively we could have allowed all tuples that were not $(0, \ldots, 0)$ and then identify any two that differ by a constant multiple in F_q.)

2. Let U and V be the set of tuples above. An edge will exist between $u \in U$ and $v \in V$ iff $u \cdot v = 0$ in the field F_q. (This is equivalent to saying that the planes which represent u and v are orthogonal to one another.)

Note 8.4 Note that the number of vertices in the graph in Definition 8.3 is $\Theta(q^{d+1})$ and the number of edges is $\Theta(q^{2d+1})$. If we denote the number of vertices by n then the number of edges is $\Theta(n^{2-\frac{1}{d}})$.

The following definition is implicit in [2].

Def 8.5 An $(\alpha n^a, \beta n^b, \chi n^c, \delta n^d)$-expander is a bipartite graph $G = (U, V, E)$ such that $|U| = |V| = n$ and the following two properties hold.

1. $(\forall Z \subseteq V)[|Z| \geq \alpha n^a \Rightarrow |\{x \in U : |N(x) \cap Z| \leq \beta n^b\}| \leq \chi n^c]$
2. $(\forall Y \subseteq V)[|Y| \geq \beta n^b \Rightarrow |N(Y) \geq n - \delta n^d]$

Alon proved the following theorem using the eigenvalues methods of [24].

Lemma 8.6 ([2]) *Let $G = (U, V, E)$ be the Geometric expander of dimension 4 over F_q. Let n be the number of vertices in U (also V). Then G is a $(3n^{3/4}, n^{1/2}, n^{1/2}, n^{3/4})$-expanding graph with $\Theta(n^{7/4})$ edges.*

The following lemma is implicitly in [2].

Lemma 8.7 *If there exist $(\alpha n^a, \beta n^b, \chi n^c, \delta n^d)$-expanders of size $O(n^e)$ then we can sort in 2 rounds using only 2-step transitive closure in $O(n^{\max\{e,d+1,c+2-a,a+1\}})$ processors.*

Theorem 8.8 ([2]) $\mathrm{sort}(2, n, 2) = O(n^{7/4})$.

Proof: This follows from Lemma 8.6 and Lemma 8.7. ▌

Empirical Note 8.9 The generation of the graph based on the tuples was accomplished by generating the tuples (which are the vertices) and then following the construction in the proof for adding edges. This method of generating graphs is very fast. Once the graph was generated, it was simply plugged into the existing code base from experimenting with Pippenger's algorithm with the transitive closure step modified to only look for direct implications.

One notable limitation of this algorithm is that it only works on certain values for n based on the values used for q and d. Due to the way in which the points are marked with the tuples, n needs to be of the form

CONDITION: $\frac{q^4-1}{q-1}$.

The proof that round one uses $O(n^{7/4})$ comparisons is easy and indicates that the number is close to $n^{7/4}$ (note the constant is 1). The proof that round two uses $O(n^{7/4})$ comparisons after doing direct implication (henceforth DI) is interesting in that it actually indicates that a weaker form of DI, which we call DM$^-$, suffices for the $O(n^{7/4})$ bound. We coded up two algorithms. Both begin by taking n and finding an $n' \geq n$ that satisfies the above condition and then applying the graph to it for round one.

Alg 1: Apply DI to the resulting directed graph. In round two make all comparisons that were not already made.

Alg 2: Apply DM$^-$ to the resulting directed graph. In round two make all comparisons that were not already made.

We made the following empirical observations.

1. Both algorithms were easy to code. The graph for round 1 is easily generated in time linear in n. (This is not surprising.)
2. It appears that in the second round Algorithm 1 made less than $0.5n^{25/16}$ comparisons and Algorithm 2 made less than $n^{13/8}$ comparisons. We caution the reader that these numbers are fitted to only four datapoints; hence, we make no claims to these being the real values the algorithm would produce for large n. However, in both cases, the quantities are far less than $O(n^{7/4})$, which is somewhat surprising. A tighter bound on round two may be provable.
3. There was more work done in round 2 in Algorithm 2 then Algorithm 1, though both were linear.

We experimented with using $d = 3$ and found that the resulting algorithm used $O(n^{5/3})$ processors per round, which is better than $O(n^{7/4})$. We also experimented with $d = 2$ and $d = 5$ and found the results to be worse. This behavior is not unexpected in retrospect, as Pippenger had discussed using $d = 3$ (as is discussed below). However, it is interesting to note that our empirical testing did lead us in that direction.

We briefly discuss Pippenger's variant on Alon's algorithm. As stated above, Alon used eigenvalue methods. In particular he showed the following.

Lemma 8.10 ([2]) *Let $H = (V, V, E)$ be a geometric expander of degree d over a field of q elements. Let $G = (V, E')$ be the graph where $(x, y) \in E'$ iff $(x, y) \in E$ and $x \neq y$. Then G has $\Theta(q^d)$ vertices and $O(q^{2-1/d})$ edges. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of the matrix for G in decreasing order. Then $\lambda_1 = \Theta(q^{2d-2})$ and $\lambda_2 = \cdots = \lambda_n = O(q^{d-1})$. The constants work out so that if $d = 3$ then $\lambda_2 \leq 2\sqrt{\lambda_1}$.*

The following lemma is implicit in [20]. It follows from Lemmas 5.7 and 8.10.

Lemma 8.11 *Let $d = 3$. Let G be as in Lemma 8.10. Let n be the number of vertices in G. Then G is an $O(n^{1/3})$-expander with $O(n^{5/3})$ edges.*

From Lemmas 8.11 and 4.7 one can easily prove Theorem 5.14.a.

Empirical Note 8.12 We coded up the $d = 3$ algorithm. Although it is supposed to use $O(n^{5/3} \log n)$ comparisons-per-round, empirically it looked like $O(n^{5/3})$. The n we tried for it may be too small to detect a difference. Both rounds seemed to use the same number of comparisons, roughly $n^{5/3}$ (note that the constant is 1).

8.1 Using Merging for $\mathrm{sort}(k, n, 2)$ with k odd

Bollobás and Thomason [9] show that $\mathrm{csort}(k, n, 1) = O(n^{\frac{3}{2} + \frac{1}{2(2^{k+1/2}-1)}})$ for k odd.

Their algorithm is similar to the approach of Haggkvist and Hell (see Section 5.1). In that algorithm you first partition the original list into sublists, recursively sort those sublists (in j rounds), and merge them back into a single ordered list (in $k - j$) rounds, where j is picked cleverly. By contrast the algorithm of Bollobás and Thomason uses $k - 2$ rounds to recursively sort the sublists and then only 2 rounds to accomplish the merging of the sublists. These last 2 rounds are done in a clever way. This uses more processors than Haggkvist-Hell; however, rather than computing the full transitive closure of the relationships learned, only 2-step transitive closure.

Theorem 8.13 ([9]) *For k odd, $\mathrm{sort}(k, n, 1) = O(n^{\frac{3}{2} + \frac{1}{2(2^{k+1/2}-1)}})$.*

Algorithm sketch: We show this by induction. If $k = 1$ then this is trivial. Assume $k > 1$ and k is odd.

1. (Preprocessing. Does not count.) Partition the n values into m sublists (each of size n/m) where $m = n^{1/2^{(k-1/2)}}$.
2. (Rounds 1 to $k-2$) Sort these sublists recursively in $k-2$ rounds.
3. (Round $k-1$) For all sublists $X = \{x_1, x_2, \ldots, x_{n/m}\}$ and all $v \notin X$ compare v to all elements in $\{x_{\sqrt{n/m}}, x_{2\sqrt{n/m}}, \ldots, x_{n/m}\}$ simultaneously. We view each sublist as having been partitioned into sublocks (e.g., the elements between $x_{4\sqrt{n/m}}$ and $x_{5\sqrt{n/m}}$ form a subblock). At the end of this round we know, for each v, which subblock it belongs to.
4. (Round k) For each v and each sublist X we know which subblock of X, v belongs. Compare v to the elements in that subblock.

A straightforward analysis shows that this algorithm uses the number of processors specified. A careful look at the last two rounds shows that it only uses 2-step transitive closure. ∎

Empirical Note 8.14 After implementing this algorithm we became curious as the whether more information was being accumulated than was being used. Specifically, after the first "merging" round, it seemed that there would potentially be enough information to eliminate the need for some later comparisons. While the presented proof offered no hints, we were curious what experimental results would show.

In the modified implementation, rather than waiting until after both of the "merging" rounds compute direct implications, we instead computed these after the first round and then asked only remaining questions during the second round. We felt that, since every value in the original list could be used as a direct intermediary between many pairs computing these direct implications, this could reduce the number of actual comparisons needed in the second merging round. Although the experimental results did show this to be true, it does not appear that we can take advantage of this by altering the number of sublists upon which we operate. While doing this can bring the processors used in the odd numbered rounds closer together, three things should be noted: (1) the odd numbered rounds all appear to be growing at the same rate, (2) most of the odd numbered rounds are already close and it is only the final round which is really lower and (3) there is no real effect towards bringing these closer to the growth rate of the number of processors used in the even numbered rounds.

Another possibility would be to increase the size of the sub-blocks within each sublist. However, this in itself would have no impact on the number of processors required in the final round.

Since this algorithm is similar to the one in Theorem 5.3 a comparison is appropriate. This algorithm gives a slightly worse upper bound than the one in Theorem 5.3; however, it does hold two possible advantages. First it requires only direct implications rather than full transitive closure. Second it works on smaller length inputs and does not have the strict limitations for the choice of n.

9 Lower Bounds

Haggkvist and Hell [14] showed that $\text{sort}(k, n) \geq \Omega(n^{1+1/k})$. Bollobas and Thomason [9] improved the constant in the $k = 2$ case. Alon and Azar [3] improved the result for all $k \geq 2$ by showing $\text{sort}(k, n) \geq \Omega(n^{1+1/k}(\log n)^{\frac{1}{k}})$. Note that this is quite close to the upper bound in Section 4.3. Of more importance, this bound is larger than $\text{rsort}(k, n)$ (see Section 6); hence sorting-in-rounds is a domain where randomized algorithms are provably better than deterministic.

Theorem 9.1 ([14]) $\text{sort}(k, n) > \frac{n^{1+1/k}}{2^{k+1}} - n/2$.

Algorithm sketch: By induction on k. For $k = 1$ this is trivial. Assume true for $k - 1$. Assume, by way of contradiction, that the first round uses $\leq 2^{-k}n^{1+1/k} - n/2$ processors. Let G be the graph of comparisons made.

By looking at the average degrees of vertices one can show that there is a set of $n/2$ nodes such that the the induced subgraph G' on them is s-colorable where $s = 2^{-k-2}n^{1/k}$. Color G' with s colors. Let $V_1, \ldots, V_s$ be the color classes. Orient G' as follows: For all $1 \leq i < j \leq s$, for all $v \in V_i$, for all $u \in V_j$, if (v, u) is an edge then set $v < u$. Orient G so that you use this orientation on G' and all the vertices not in G' are less than all the vertices in G'. Once the transitive closure of G is taken one still needs to sort each V_i in $k-1$ rounds (one also needs to sort those elements not in G' but this is not needed for the lower bound). One can show the lower bound by using the inductive lower bound on each V_i and some algebra. ∎

Theorem 9.2 ([9]) *For all* $c < \sqrt{3/2}$, $\text{sort}(2, n) \geq cn^{3/2}$

Algorithm sketch: This proof is similar to that of Theorem 9.1 except that the $V_1, \ldots, V_s$ are obtained by a greedy coloring and more care is taken in showing the largest value of $\sum_{i=1}^{s} \text{sort}(k - 1, |V_i|)$. Lagrange multipliers are used. ∎

The key to the proofs of Theorems 9.1 and 9.2 is that we still need to sort each V_i. The proof does not use the fact that you might have to make some comparisons between vertices in different V_i. To improve this lower bound Alon and Azar showed that you will have to make such comparisons.

Lemma 9.3 **([3])** *Let G be a graph with n vertices and dn edges. There exists an induced subgraph on $\frac{n}{4}$ vertices such that (1) G' has degree $< 4d$, and (2) there is a $4d$-coloring of G' with color classes $V_1, \ldots, V_{4d}$ such that for all $1 \le i, j \le 4d$, for all $x \in V_i$, there are at most $2^{|i-j|+1}$ neighbors in V_j.*

Algorithm sketch: Remove successively the highest degree vertex $\frac{n}{2}$ times. Let G' be the induced subgraph on the remaining $\frac{n}{2}$ vertices. One can show that G' has degree $< 4d$. Clearly G' is $4d$-colorable. Let $U_1, \ldots, U_{4d}$ be the color classes. A probabilistic argument shows that there exists a permutation of $\{1, \ldots, 4d\}$ that satisfies the properties needed. ▮

Lemma 9.4 **([3])** *Let d be such that $d = o(n)$ and $d = \Omega(\log n)$. Let G be a graph with n vertices and dn edges. There exists an orientation of G such that the complement of its transitive closure has at least $\Omega(\frac{n^2}{d}\log(\frac{n}{d}))$ edges.*

Algorithm sketch: Use Lemma 9.3 to obtain G' and $V_1, \ldots, V_{4d}$ as specified there. Orient G' as follows: For all $1 \le i < j \le m$, for all $v \in V_i$, for all $u \in V_j$, if (v, u) is an edge then set $v < u$. Orient G so that you use this orientation on G' and all the vertices not in G' are less than all the vertices in G'. The complement of the transitive closure will not contain any edges within an V_i. In addition, because of the limit on how many edges can go from an V_i to an V_j, one can estimate additional lower bounds on the number of edges in the complement of the transitive closure. (This is highly non-trivial.) ▮

Theorem 9.5 **([3])** *For $k \ge 2$, $\mathrm{sort}(k, n) \ge \Omega(n^{1+1/k}(\log n)^{1/k})$.*

Algorithm sketch:

We prove this by induction. Note that $k = 2$ is the base case and is non-trivial. Assume that there is an algorithm that sorts n elements in 2 rounds and takes dn processors. We know that $d = \Omega(n^{1/3})$ by Theorem 9.1. We can assume $d = o(n^{2/3})$ since if it is not then the theorem for $k = 2$ is already true. Let G be the graph representing the first round. Because of the bounds on d we can apply Lemma 9.4 to the graph to obtain an orientation such that the complement of the transitive closure has $\Omega(\frac{n^2}{d}\log(\frac{n}{d}))$ edges. Hence the second round needs $\Omega(\frac{n^2}{d}\log(\frac{n}{d}))$ processors. Algebra shows that $d = \Omega(\sqrt{n \log n})$.

We now sketch the induction step. It will be *easier* than the $k = 2$ case since it does not use Lemma 9.4. Assume the lower bound for $k - 1$ where $k \ge 3$. Assume there is an algorithm for sorting in k rounds. Let G be the graph representing the first round of the algorithm. Assume G has dn edges. By Turan's theorem (see [5] for a nice probabilistic proof) a graph with dn edges has an independent set of size $\frac{n}{2d+1}$. By repeated application of Turan's theorem we can find $s = \Omega(d)$ pairwise disjoint independent sets of size $\Omega(\frac{n}{1+d})$ which we denote $V_1, \ldots, V_s$. Let V_0 be all the other vertices. Orient G as follows: For all $1 \le i < j \le s$, for all $v \in V_i$, for all $u \in V_j$, if (v,u) is an edge then set $v < u$. The remaining $k-1$ rounds need to sort each V_i, $i \ge 1$. Algebra and the induction hypothesis suffice to prove the result. ▮

10 Open Problems

The next section has tables of known results, both upper and lower bounds. The tables yield many open questions about closing these gaps.

There are no lower bounds for constructive sorting except those bounds that come from general sorting. Hence another open question would be to either obtain lower bounds for csort(k,n) that use the fact that the algorithm is constructive, or show that any sorting algorithm can be turned into a constructive one.

Another open problem is to obtain simpler proofs of the known upper bounds, especially the constructive ones.

11 Summary of Results

In this section we put all the known results into tables. We leave out the big-O's and big-Ω's unless there is an interesting point to be made about the constants.

11.1 Nonconstructive Methods for Sorting

k	$k=2$	$k=3$	Ref
$n^{(3\cdot 2^{k-1}-1)/(2^k-1)} \log n$	$n^{5/3}\log n$	$n^{11/3}\log n$	[15]
	$n^{3/2}\log n$		[9]
$n^{1+1/k}(\log n)^{2-2/k}$	$n^{3/2}\log n$	$n^{4/3}(\log n)^{4/3}$	[20]
	$n^{3/2}\frac{\log n}{\sqrt{\log\log n}}$		[3]
$n^{1+1/k}\frac{(\log n)^{2-2/k}}{(\log\log n)^{1-1/k}}$	$n^{3/2}\frac{\log n}{\sqrt{\log\log n}}$	$n^{4/3}\frac{(\log n)^{4/3}}{(\log\log n)^{2/3}}$	[6]

1. All of the above results use the Probabilistic method.
2. In [15] a graph is picked at random from the set of all graphs with n vertices and n^{α_k} edges where $\alpha_k = \frac{3\cdot 2^{k-1}-1)}{(2^k-1)}$.

3. In all of the other algorithms a graph was picked by assigning to each edge a probability.
4. The $n^{1+1/k}(\log n)^{2-2/k}$ algorithm was fairly easy to code and behaved as expected with low implicit constants.

11.2 Constructive Methods for Sorting

k	$k=2$	$k=4$	Ref
	$\frac{39}{45}\binom{n}{2}$		[14]
	$\frac{4}{5}\binom{n}{2}$		[8]
$n^{1+(2/\sqrt{2k})}$	n^2	$n^{20/13}$	[15]
$n^{1+(2/\sqrt{2k})}$	$n^{7/4}$	$n^{26/17}$	[2]
$n^{1+(2/\sqrt{2k})}$	$n^{7/4}$	$n^{3/2}$	[13, 20]
$n^{1+2/(k+1)}(\log n)^{2-4/(k+1)}$	$n^{5/3}(\log n)^{2/3}$	$n^{7/5}(\log n)^{6/5}$	[17, 20]
$n^{1+1/k+o(1)}$	$n^{3/2+o(1)}$	$n^{5/4+o(1)}$	[18, 26]

1. The first two results listed for general k are both approximations for a general recurrence.
2. All of the $n^{1+(2/\sqrt{2k})}$ algorithms are recursive and were hard to code. The first one used a trivial base case and only worked for large n. The rest used more sophisticated base cases and worked well for small n. All of them had quite reasonable multiplicative constants; usually less than 1.
3. The [17, 20] algorithms were based on certain types of graphs that can be generated constructively but are difficult to deal with. We did not code it up.
4. The [18, 26] algorithm would only make sense for n that were quite large. Hence we did not code it up.

11.3 Limited Closure Sorting

The only result here are for 2-round sorting.

d	$d=2$	Ref	Constructive?
$\frac{1}{2d}n^{1+\frac{d}{2d-1}}(\log n)^{1/2d-1}$	$\frac{1}{4}n^{5/3}(\log n)^{1/3}$	[9]	No
	$n^{7/4}$	[2]	Yes

The first algorithm relies on picking a graph at random in a way that is hard to code up, so we did not code it up. The second algorithm is easy to code up and when we did so it used $n^{7/4}$ comparisons in the first round and less in the second.

11.4 Lower Bounds

We include these known lower bounds for completeness.

Problem	Bound	Ref	Comments
sort(k,n)	$\frac{n^{1+1/k}}{2^{k+1}} - n/2$	[14]	
sort(k,n)	$(\sqrt{3/2}-\epsilon)n^{1+1/k}$	[9]	$k \ge 2$
sort(k,n)	$n^{1+1/k}(\log n)^{\frac{1}{k}}$	[3]	$k \ge 2$
sort$(2,n,2)$	$n^{5/3}$	[9]	
sort$(2,n,d)$	$n^{1+d/2d-1}$	[9]	
Merging	$n^{\frac{2^k}{2^k-1}}$	[15]	

11.5 Comparisons Between Different Algorithms

In this section we compare the various algorithms to each other. To compare two (say) 3-round algorithms we take as a measure the maximum number of comparisons used in a round. For example, if a run of the algorithm used 14 comparisons in the first round, 18 in the second round, and 11 in the third round, then we would say '18 comparisons.'

We comapre the following pairs of algorithms.

1. Pippenger's algorithm with Haggkvist and Hell's algorithm (henceforth HH) for both 3-round and 4-round.
2. Pippenger's algorithm with Alon's algorithm for 2-rounds. (Alon's only works for 2 rounds.)
3. Pippenger's algorithm with Alon-Azar-Vishkin's algorithm (henceforth AAV) for 3 and 4 rounds. The AAV is randomized; however, we compare what it does in its worse case.

Pippenger's algorithm performs better than HH or Alon's. This underscores the point that nonconstructive algorithms can outperform constructive ones. AAV beats Pippenger, which shows that randomized is better still. The drawback of randomized algorithms is that if you are unlucky in a particular run it may take longer; however, this did not happen in our

Figure 14

Figure 15

Figure 16

observations. Note also that the drawback of nonconstructive algorithms is that if you are unlucky in the preprocessing then you may be stuck with an algorithm that is not good on any run; however, this did not happen. Hence we are comparing Pippenger's algorithm to AAV's algorithm when neither one experienced bad luck.

We note that comparing Pippenger's algorithm to Alon's in terms of number of comparisons is unfair since Alon's algorithm has the benefit of limited transitive closure.

In Figures 14 and 15 we compare Pippenger's algorithm with HH's algorithm. For the most part all the algorithms have the same shape as their analytic bound, resulting in Pippenger's doing better. We again note the one exception— HH does particularly badly in 4-rounds before $n = 8192$. This is because the algorithm assumes that both n and the blocksize are powers of 2. This assumption affects the algorithm in many ways resulting in the bad runtime mentioned.

In Figure 16 we compare Alon's 2-round constructive algorithm with Pippenger's 2-round nonconstructive algorithm. Both algorithms have the same shape as their analytic bound, resulting in Pippenger's doing better.

Figure 17 Figure 18

In Figures 17 and 18 we compare Pippenger's algorithm with AAV's algorithm. For the most part all the algorithms have the same shape as their analytic bound, resulting in AAV doing slightly better.

12 Acknowledgment

We thank Nick Pippenger for intellectual inspiration and helpful discussions.

References

[1] S. Akl. *Parallel computation: models and methods.* Prentice Hall, 1997.

[2] N. Alon. Eigenvalues, geometric expanders, sorting in rounds, and Ramsey theory. *Combinatorica*, 6, 1986.

[3] N. Alon and Y. Azar. Sorting, approximate sorting, and searching in rounds. *SIAM Journal on Discrete Mathematics*, 1:269–280, 1988.

[4] N. Alon, Y. Azar, and U. Vishkin. Tight bounds for parallel comparison sorting. In *Proc. of the 29th IEEE Sym. on Found. of Comp. Sci.*, pages 502–510, 1988.

[5] N. Alon and J. Spencer. *The Probabilistic Method.* Wiley, 1992.

[6] B. Bollobás. Sorting in rounds. *Discrete Mathematics*, 72, 1988.

[7] B. Bollobás and P. Hell. Sorting and graphs. In I. Rival, editor, *Graphs and Orders*, pages 169–184. D. Reidel Publishing Company, 1985.

[8] B. Bollobás and M. Rosenfeld. Sorting in one round. *Israel Journal of Mathematics*, 38, 1981.

[9] B. Bollobás and A. Thomason. Parallel sorting. *Discrete Applied Mathematics*, 6, 1983.

[10] H. Davenport. *Multiplicative number theory.* Springer, 1980.

[11] P. Erdos and A. Renyi. On a problem in the theory of graphs (in Hungarian). *Publ. Math. Inst. Hungar. Acad. Sci.*, 7, 1962.

[12] W. Gasarch, E. Golub, and C. Kruskal. A survey of constant round parallel sorting. *Bulletin of the European association of theoretical computer science(BEATCS)*, 72:84–102, 2000.

[13] E. Golub. *Empirical Studies in Parallel Sorting, 1999.* PhD thesis, U. of MD, College Park, 1999. `www.cs.umd.edu/~egolub/dissert.pdf`.

[14] R. Haggkvist and P. Hell. Parallel sorting with constant time for comparisons. *SIAM J. Comput.*, 10(3):465–472, 1981.

[15] R. Haggkvist and P. Hell. Sorting and merging in rounds. *SIAM Journal on Algebraic and Discrete Methods*, 3, 1982.

[16] F. T. Leighton. *Introduction to parallel algorithms and architectures: arrays, trees, and hypercubes.* Morgan Kaufman, 1992.

[17] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 1988.

[18] N. Nisan and D. Zuckerman. Randomness is linear in space. *Journal of Computer and Systems Sciences*, 52(1):43–52, Feb. 1996. Earlier version in FOCS93. The title in FOCS93 was *More deterministic simulations in Logspace.*

[19] A. W. Omer Reingold, Ronen Shaltiel. Extracting randomness via repeated condensing. Technical Report TR 00-059, Electronic Colloquium on Computational Complexity (ECCC), 2000. See www.eccc.uni-trier.de/eccc/.

[20] N. Pippenger. Sorting and selecting in rounds. *SIAM J. Comput.*, 16:1032–1038, 1987.

[21] N. Pippenger. Personal communication, 2000.

[22] S. P. V. Ran Raz, Omer Reingold. Extracting all the randomness and reducing the error in trevisan's extractors. In *Proc. of the 31th ACM Sym. on Theory of Computing*, 1999.

[23] J. Spencer. *Ten Lectures on the Probabilistic Method.* Conf. Board of the Math. Sciences, Regional Conf. Series, AMS and MAA, 1987.

[24] M. Tanner. Explicit construction of concentrators from generalized n-gons. *SIAM Journal on Algebraic and Discrete Methods*, 5, 1984. Despite the title the main result relates eigenvalues to expanders.

[25] L. G. Valiant. Parallelism in comparison problems. *SIAM J. Comput.*, 4(3):348–355, Sept. 1975.

[26] A. Wigderson and D. Zuckerman. Expanders that beat the eigenvalue bound: explicit construction and applications. *Combinatorica*, 19:125–138, 1999. Earlier version appeared in STOC93.