


Sorting in One and Two Rounds Using t -Comparators

Ran Gelles ✉ 

Bar-Ilan University, Ramat Gan, Israel

Zvi Lotker ✉ 

Bar-Ilan University, Ramat Gan, Israel

Frederik Mallmann-Trenn ✉ 

King's College London, UK

Abstract

We examine sorting algorithms for n elements whose basic operation is comparing t elements simultaneously (a t -comparator). We focus on algorithms that use only a single round or two rounds – comparisons performed in the second round depend on the outcomes of the first round comparators. Algorithms with a small number of rounds are well-suited to distributed settings in which communication rounds are costly.

We design deterministic and randomized algorithms. In the deterministic case, we show an interesting relation to design theory (namely, to 2-Steiner systems), which yields a single-round optimal algorithm for $n = t^{2^k}$ with any $k \geq 1$ and a variety of possible values of t . For some values of t , however, no algorithm can reach the optimal (information-theoretic) bound on the number of comparators. For this case (and any other n and t), we show an algorithm that uses at most three times as many comparators as the theoretical bound.

We also design a randomized Las-Vegas two-round sorting algorithm for any n and t . Our algorithm uses an asymptotically optimal number of $O(\max(\frac{n^{3/2}}{t^2}, \frac{n}{t}))$ comparators, with high probability, i.e., with probability at least $1 - 1/n$. The analysis of this algorithm involves the gradual unveiling of randomness, using a novel technique which we coin the *binary tree of deferred randomness*.

2012 ACM Subject Classification Theory of computation → Sorting and searching; Mathematics of computing → Probabilistic algorithms; Theory of computation → Distributed algorithms

Keywords and phrases Sorting, Steiner-System, Round Complexity, Deferred Randomness

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.27

Related Version *The full version of this work is available in:* [arXiv:2405.12678](https://arxiv.org/abs/2405.12678) [18]

Funding *Ran Gelles:* Research supported in part by the United States-Israel Binational Science Foundation (BSF) through Grant No. 2020277.

Frederik Mallmann-Trenn: Was funded by the EPSRC grant EP/W005573/1.

Acknowledgements R. Gelles would like to thank Paderborn University and CISPA – Helmholtz Center for Information Security for hosting him while part of this research was done. The authors would also like to thank the anonymous reviewers for multiple helpful comments.

1 Introduction

Sorting has been a fundamental task for computers (and earlier electronic devices) since the inception of computer history [24, 13]. Many sorting algorithms are *comparison-based*, meaning that there exists some device that compares pairs of elements and decides which of them is the larger. By comparing multiple pairs, one can obtain a full order of all elements. It is well known that if pairs are being compared, $\Theta(n \log n)$ comparisons are needed in order to fully sort any possible set of n elements. Such sorting, however, assumes one can apply



© Ran Gelles, Zvi Lotker, and Frederik Mallmann-Trenn;
licensed under Creative Commons License CC-BY 4.0
38th International Symposium on Distributed Computing (DISC 2024).
Editor: Dan Alistarh; Article No. 27; pp. 27:1–27:20



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

comparisons in an adaptive manner, i.e., one can determine which pairs of elements to compare next based on results of previous comparisons. It is not too difficult to see that without this adaptive selection of elements, $\Omega(n^2)$ comparisons are needed (see also Lemma 4 below).

In contrast to general-purpose CPUs, which allow fast comparison of two elements, specialized hardware that can be found in system-on-a-chip systems and GPUs, allows comparing larger sets of elements. Motivated by the above, in this work we explore sorting algorithms that use t -comparators. These blocks allow t elements to be compared simultaneously to determine their total order, rather than comparing them in pairs. Our initial focus is on deterministic, non-adaptive sorting algorithms where all comparisons are pre-determined and independent of prior outcomes. Additionally, we consider randomized algorithms with a limited degree of adaptiveness. In particular, we design sorting algorithms with two rounds, where the second round can use the comparison outcomes from the first round. In both cases, our goal is to minimize the number of t -comparators used.

To further motivate the case of sorting with t -comparators (t -sorting) in a single round, consider the following scenario, which is very common in the Computer Science community. A conference program committee (PC) is set to decide on the ranking of the n submitted papers. Let us assume that there is an “absolute truth”, namely, that there exists a total ordering of the papers, and that each PC member outputs the “true” ordering of any number of papers assigned to them.¹ To balance out the load, the papers are split so that each PC member receives t papers. Note that the same paper can be sent to multiple PC members. Each PC member, individually, returns to the chair the total order of the set of papers assigned to them. The chair collects all these outputs and composes a total ordering of the n papers, that is consistent with all the partial sets. Assume we wish the chair’s output to be the “true” ordering of the papers, how many PC members are needed, as a function of n and t ? Note that the chair assigns the papers once, without having any information about papers, that is, this is a non-adaptive t -sorting with a single round.

1.1 Deterministic Sorting

Consider deterministic t -sorting algorithms with a single round. Similar to the case of $t = 2$, that requires comparing all $\binom{n}{2}$ possible pairs, it can easily be shown that for any t , at least $\gamma_{n,t} = \binom{n}{2} / \binom{t}{2}$ many t -comparators are needed in order to fully sort n elements. This stems from the fact that in order to learn the total ordering of n elements, we need to learn the relative-order of all $\binom{n}{2}$ pairs, while each t -comparator gives us information about at most $\binom{t}{2}$ different pairs of elements (Corollary 5).

Our first question is whether this bound is achievable, that is, whether there exists a single round t -sorting algorithm that utilizes exactly $\gamma_{n,t}$ comparators. We first show a way to perform t -sorting with at most $3\gamma_{n,t}$ comparators (Lemma 6). The idea is rather simple: we divide the elements into disjoint subsets, where each subset contains $t/2$ different elements. Then, we go through all possible pairs of subsets, and for each such pair we compare the t elements of their union using a separate t -comparator. This guarantees that any two elements are compared by at least one comparator, so a total-ordering of the n elements can be deduced from the results of the $\binom{\lceil n/(t/2) \rceil}{2} < 3\gamma_{n,t}$ different comparisons.

Our main result is an algorithm with an optimal level of $\gamma_{n,t}$ t -comparators for the case where t is a power of a prime and $n = t^{2^k}$, for any positive integer $k \in \mathbb{N}$. Namely,

¹ We realize that, in real life, no such absolute truth exists, and that PC members are heavily biased, etc. These extensions make a very interesting direction for followup questions. We briefly discuss future directions in Section 1.5.

► **Theorem 1** (main, deterministic). *Let t be a power of a prime and let $n = t^{2^k}$, $k \in \mathbb{N}$. Then, there exists a deterministic single-round, t -sorting algorithm that utilizes exactly $\binom{n}{2} / \binom{t}{2}$ comparators.*

In order to obtain the above optimal sorting, we show a connection between sorting and combinatorial design theory. Consider the case where t is a prime power and $k = 1$, that is, $n = t^2$, a setting that attained a lot of interest in the past, especially by hardware-implementation oriented designs [40, 35, 36]. We essentially show that sorting with $\gamma_{n,t}$ comparators is equivalent to an *Affine Plane* of order t . An affine plane (see e.g., [22, 31]) is a design structure composed of elements (“points”) and subset of elements (“lines”) that guarantees the following properties: (P1) every two points belong to a *unique* line, (P2) every line contains at least two points, and (P3) not all points are co-linear. Further, it satisfies the Euclidean Property (A1): for every line L and any point p outside L , there exists a unique line that contains p and is parallel to L . It is known that all lines in an affine plane contain exactly the same number of points; call this number *the order of the plane*. It is also known that an affine plane of order t contains $t^2 + t$ lines.

If we think about points as the elements we wish to sort and about lines as subsets of t points which we compare via a single comparator, finding an affine plane of order t provides the property that any two elements are being compared *exactly* a single time, i.e., by a single comparator, leading to the optimal bound of $\gamma_{n,t}$ comparators.

An affine plane of order t is easy to construct for any t that is a power of a prime. Let \mathbb{F} be a finite field with t elements, and consider pairs of elements (x, y) , i.e., the plane \mathbb{F}^2 . In this plane, any two points (x_1, y_1) and (x_2, y_2) , define a unique line that passes through them, namely $y = \frac{y_1 - y_2}{x_1 - x_2}x + \frac{y_2x_1 - y_1x_2}{x_1 - x_2}$ if $x_1 \neq x_2$ and the line $\{(x_1, y) \mid y \in \mathbb{F}\}$, otherwise. It is easy to verify this structure satisfies all the properties of an affine plane (see [31, Section 3.2]).

Affine planes are a special case of a more general combinatorial structure known as Steiner systems (Definition 10). Indeed, if we change assumption (A1) so that there exists no parallel lines at all (also known as the Elliptic Property), but still require that any two points define a unique line, we would still get a sorting algorithm in which any two elements are being compared against each other exactly once. In this case, the resulting structure is again a special case of a Steiner system known as a *Projective Plane*. Known constructions of projective planes imply that for any $t - 1$ being a power of a prime, one can sort $t^2 - t + 1$ elements using exactly $t^2 - t + 1$ many t -comparators, where every pair of elements is being compared exactly once. These two constructions are summarized as Theorem 12.

We lift the above result from optimally sorting t^2 elements to optimally sorting t^{2^k} , by developing a composition theorem (Lemma 15) that recursively performs sorting of t^{2^k} elements by utilizing an optimal number of $t^{2^{k-1}}$ -comparators, for any $k > 1$.

1.2 Randomized Sorting

Similar to the deterministic case, if one does not bound the number of adaptive rounds a randomized sorting algorithm is allowed to make, optimal sorting can easily be achieved. For instance, Beigel and Gill [8] showed a generalized t -quicksort algorithm that sorts n elements by utilizing at most $4 \frac{n \log n}{t \log t}$ many t -comparators, which is optimal, maybe up to the constant (see Theorem 16). However, this algorithm requires $O(\log_t n)$ adaptive rounds. Indeed, recall that quicksort works in rounds, where at each round the algorithm selects (one or more) pivot elements. These elements are used to “bucket” the rest of the elements into disjoint subsets, meaning that all elements greater than one pivot and less than the next pivot belong

27:4 Sorting in One and Two Rounds Using t -Comparators

to the same bucket. Then, each such bucket is recursively sorted by the same method. Since each round depends on the pivots and buckets of the previous rounds, $O(\log_t n)$ recursive rounds are needed [8].

Our second question in this work is how to obtain optimal randomized t -sorting algorithms with restricted number of rounds. Since we already analyzed the case of a single round and reached optimal results, in the second part of this work we address the case of *two* rounds. Our goal is to minimize the number of t -comparators used to sort n elements in a Las-Vegas algorithm, where the output is correct with probability 1 but the *number of comparators* used is a random variable that varies between different instances.

Our main result for this part is as follows.

► **Theorem 2** (main, randomized). *Let $t < n$ be given. There exists a (Las-Vegas) randomized sorting algorithm for n elements with two rounds, that utilizes $O\left(\max\left(\frac{n^{3/2}}{t^2}, \frac{n}{t}\right)\right)$ many t -comparators, with probability at least $1 - 1/n$.*

We note that for the case where $n = t^2$, our algorithm uses $O(t)$ comparators which is asymptotically optimal since $\frac{n \log n}{t \log t} = \Theta(t)$. We further note that a result by Alon and Azar [3] implies that the expected number of comparators used in our algorithm when $t \leq \sqrt{n}$, is also tight.

The high-level idea of the two-round algorithm is to perform a single round of “quicksort” and then to optimally (deterministically) sort each resulting bucket, rather than recursively sorting it. In more details, let m be some fixed parameter. Our algorithm starts by sampling m elements that will serve as pivots. We bucket all the elements by dividing the rest $n - m$ elements into subsets of size m elements each, and comparing each such subset, along with the m pivots by utilizing at most $3\gamma_{2m,t}$ many t -comparators (per subset). This step tells us, for each one of the $n - m$ elements, between which two pivots it resides.

A pseudo code of our 2-round randomized algorithm is given below as Algorithm 1 for the case $t \leq m$. The case $t > m$ is very similar and is covered in Section 4.

■ **Algorithm 1** A randomized 2-round sorting for any n, t with $t \leq m$.

Round 1:

- 1: Let P be a set of m elements from A , each sampled uniformly and independently from A .
- 2: Partition $A \setminus P$ into subsets A_1, \dots, A_k of size at most m each. ▷ $k = \lceil (n - m)/m \rceil$
- 3: **for all** $i \in [k]$ **do**
- 4: Sort $P \cup A_i$ using the optimal 1-round deterministic algorithm.
- 5: **end for**

Round 2:

- 6: Let $P = (p_1, \dots, p_m)$ be the ordered elements in P . For $1 \leq i \leq m - 1$, set S_i to contain all the elements which are greater than p_i but lower than p_{i+1} . Set S_0 to be all the elements lower than p_1 and S_m be all the elements greater than p_m .
 - 7: **for all** $0 \leq i \leq m$ **do**
 - 8: Sort S_i using the optimal 1-round deterministic algorithm.
 - 9: **end for**
-

In expectation, each bucket is of size $\approx n/m$ and sorting a bucket of this size takes $3\gamma_{n/m,t}$ many t -comparators. If all buckets had size exactly n/m , this would lead immediately to the desired result of $3\frac{n}{m}\gamma_{2m,t} + 3(m+1)\gamma_{n/m,t} = O\left(\frac{nm}{t^2} + \frac{n^2}{mt^2}\right)$. This quantity is minimal when

$m \approx \sqrt{n}$ (ignoring constants), leading to the claimed $O(\frac{n^{3/2}}{t^2})$ bound.² Unfortunately, buckets' sizes vary, and some of them might be much larger, say, of size $(n/m) \log(n/m)$. However, our analysis shows that this event is very rare and the additional number of comparators needed to handle these cases is rather small. More specifically, in our analysis, we formulate a balls-into-bins process to distribute elements into buckets, and bound the number of such bad events using the balls-into-bins process. Let us now expand on the techniques used in this analysis.

1.2.1 Techniques: The binary tree of deferred randomness

Let us start by describing the balls-into-bins process we use. Consider the n elements, and rename them a_1, \dots, a_n so that they are sorted. Starting with a_1 , we group together sequences of cn/m consecutive elements, for some sufficiently large constant c . We call each such group a *bin*; namely, the first bin is $b_1 = \{a_1, \dots, a_{cn/m}\}$ the second bin is $b_2 = \{a_{cn/m+1}, \dots, a_{2cn/m}\}$ and so on, resulting in a total of m/c bins overall. The balls will be the m elements we pick as pivots. That is, let $P = \{p_1, p_2, \dots, p_m\}$ be the elements selected as pivots. Since each pivot is sampled uniformly at random, the selection of some p_i is equivalent to throwing a ball to bin b_j where $p_i \in b_j$.³

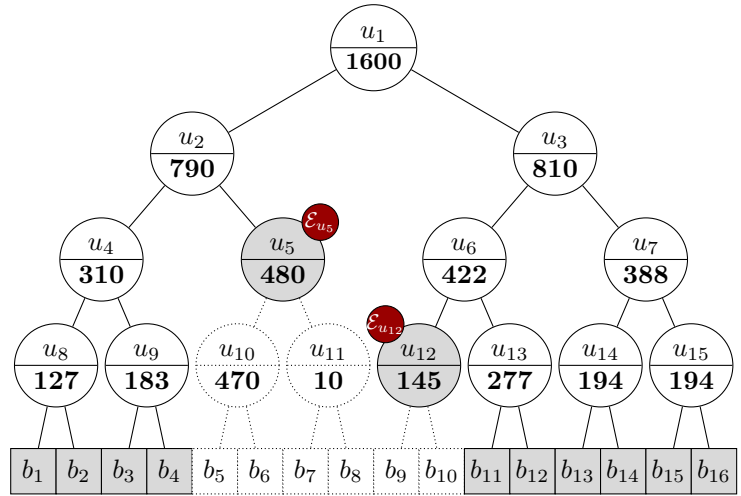
If each bin has a ball, then each “bucket” has at most $2cn/m$ elements, and the cost, measured in the number of comparators needed to sort that bucket, is as desired. However, the absence of a ball in a bin implies larger buckets. That is, the size of the bucket, and hence the cost of sorting it, is determined by the stretch of bins without balls (up to two additional bins, one from each side). In other words, in order to bound the cost of the second round, we throw $|P| = m$ balls uniformly at random into m/c bins and count the length of consecutive *empty* bins. Recall that $m = \sqrt{n}$; we will substitute this value to avoid cumbersome equations in the following.

A straightforward balls-into-bins analysis shows that there are c pivots per bin in expectation and that the probability of not having a pivot in c' consecutive bins scales as $e^{-\Omega(c')}$. Ideally we would like to use the above probability and obtain a polynomially-small failure probability by considering all the bins at the same time. Unfortunately, this approach breaks due to the correlation between empty bins. Indeed, the fact that some bins are empty indicates that the balls went somewhere else, altering the probability of having empty bins elsewhere. The bins' loads are negatively correlated. This means that concentration bounds could potentially be used for negatively correlated variables. However, there are many obstacles to this approach. First, note that while the loads of the bins are negatively correlated, we actually need to bound different variables, namely, the lengths of consecutive sequences of empty bin. Second, defining these variables and analyzing their probability function, as well as proving that they are negatively correlated, seems to be a difficult task. Finally, note that even the number of these random variables, is itself a random variable.

Instead, we introduce the concept of a *binary tree of deferred randomness* that enables a more straightforward analysis of the concentration of empty bins, circumventing difficulties arising from their dependencies.

² The term $O(n/t)$ in Theorem 2 stems from the other case, where $t > m$, i.e., $t > \sqrt{n}$.

³ We note that this balls-into-bin process differs slightly from our pivot selection process in the sense that it samples pivots with replacement, while the original process samples without replacement. However, one could modify the original process by allowing the same element to be sampled multiple times, and later ignore these extra copies. It is immediate that sampling without replacement can only create smaller bins and thus improve the overall complexity.



■ **Figure 1** The figure shows the distribution of pivots (balls) on the *tree of deferred randomness*, marked as the numbers in each node. Here we have $\sqrt{n} = 1600$ pivots and 16 bins ($c = 100$). In the first two levels, the distribution is about even. The node u_{11} receives too few balls and so the event \mathcal{E}_{u_5} holds. Similarly, b_9 gets too few balls (bin's balls are not shown in the figure), causing $\mathcal{E}_{u_{12}}$ to happen. The nodes in gray portray the set END described in detail in the full version.

We think of the assignment of a pivot (a ball) to a bin as the bit-string describing the bin where the pivot ends, that we reveal bit-by-bit. We define a binary tree, where each one of the \sqrt{n}/c bins is a leaf. Thus, the tree has a depth of $\log(\sqrt{n}/c)$ (assuming \sqrt{n}/c is a power of two). We define the following iterative process of assigning balls to the leaves of the tree: Initially we have \sqrt{n} balls at the root. At every step, at every node u , we randomly assign each ball to one of u 's children. This is equivalent to revealing the next bit in the string representing the bin to which the pivot belongs to. The advantage of this approach lies in the careful revelation of the randomness. At every level, we can derive concentration bounds without affecting the following levels – the only thing that matters at a given node is how many balls arrive at it.

Consider the binary tree of deferred randomness after all balls are assigned and follow an arbitrary path from the root to a leaf v . There are two cases. In the ideal case, at every node along the path to v , the number of balls going left and right is close to the expected value, namely, close to half. If this happens, then enough balls propagate along this path and with high probability at least one of them will reach the leaf v . This is the good scenario, since if this holds for many bins, the cost of sorting their elements will be very close to the expected cost.

The second case is when the concentration fails at some node u on the path, and the assignments of the balls is not close to half. If this happens first at node u , we say that the bad event \mathcal{E}_u occurred, stop the process there (i.e., ignore other nodes in u 's subtree), and charge a cost as if only a single ball reaches the bins under the node u . In other words, if there are ℓ balls at node u , we assume that all the ℓ pivot selections ended up picking the same element. By doing so, we overestimate the size of the resulted bucket to contain all the elements in all the bins below u . Specifically, we charge this event with the cost of sorting $O(\ell \cdot c\sqrt{n}) > |\text{bins}(u)| \cdot c\sqrt{n}$ elements; here we use the fact that, as long as the bad event \mathcal{E}_u does not happen, the number of balls reaching u always exceeds the number of bins in the subtree of u , $|\text{bins}(u)|$.

Figure 1 illustrates the infiltration of balls through the tree: a node u at level i is associated to the $2^{\log(c\sqrt{n})-i}$ bins below it. The number inside a node denotes how many balls are assigned to that node. When u assigns the balls to its children, each ball picks one

of the children uniformly at random, so each of the children is assigned half of u 's balls, in expectation. The process continues until we reach the leaves at level $\log(c\sqrt{n})$. In the rare event that balls are distributed in a very skewed manner, the bad event \mathcal{E}_u happens. For instance, while u_5 has 480 balls, they split very unevenly among its children, causing the bad event \mathcal{E}_{u_5} . The process stops there, i.e., we do not care how the balls continue in the subtree of u_5 and in particular, \mathcal{E}_u never happens in any of u 's descendants. Since \mathcal{E}_{u_5} happens and the process stops there, the analysis charges an *amortized* cost which is proportional to sorting a bucket of size of 4 bins (due to the 4 bins b_5, b_6, b_7, b_8 – for all we know, all the balls could end up in b_8 , creating a single bucket that consists all the respective elements). In fact, we upper bound this cost by the number of balls that arrive to u_5 , whose expectation in this example is $4c \gg 4$. The situation might get even worse, since \mathcal{E}_{12} occurs as well. This effectively means that a single bucket might consist of all the element in bins b_4 – b_{10} . The dependency between neighboring nodes with bad events complicates the cost analysis. However, by summing up the costs of all these events, we can derive the amortized cost per such bad event and simplify the analysis by considering a single event at a time.

Luckily, the higher up in the tree a node is, the more balls the node holds and the less likely the concentration bound will fail. The lower in the tree the node is, the lower the cost is. In particular, once we approach the lower levels of the tree, the bad event \mathcal{E}_u occurs with constant probability. This does not pose any trouble, because the cost in this case is only a constant factor larger than the expected cost of the case where each bin has at least one ball in it. Overall, we show that for every level of the tree, the cost imposed in our process is very close to its expectation, with high probability (at least $1 - 1/n^2$). Taking a union bound over all the (at most n) levels of the our tree of deferred randomness yields the desired claim. We give the full details in Section 4.3.

1.3 Related Work

A fundamental task like sorting naturally attracted a lot of attention in numerous variants and settings. To put our result in the right context, in this section we mention just a few of these variants and we mainly focus on *comparison-based* sorting algorithms. We refer the reader to surveys [27, 16, 38] and books [24, 13, 2] for a more complete treatment on the background of (general) sorting.

The task of sorting in small number of rounds was initiated by the work of Häggkvist and Hell [20], who considered the case of sorting n elements in a single round by comparing pairs of elements (i.e., $t = 2$). While they do not give any explicit sorting algorithm, they bound the number of 2-comparators required for sorting in d -rounds by $\Omega(n^{1+1/d})$ from below and by $O(n^{\alpha_d \log n})$ from above, for a constant α_d that monotonously decreases towards $3/2$ as d grows. Specifically, for $d = 2$, they prove that the optimal number of comparisons lies within the range $(C_1 n^{3/2}, C_2 n^{5/3} \log n)$ for some constants C_1, C_2 . Alon, Azar, and Vishkin [5] improved the lower bound to $\Omega(n^{1+1/d}(\log n)^{1/d})$. Alon and Azar [3, 4] lower-bounded the *average* number of comparisons by $\Omega(dn^{1+1/d})$, for any d -round algorithm with $d \leq \log n$. They also improved the upper bound to $O(n^{1+1/d} \log n)$ for a fixed d , and to $dn^{1+O(1)/d}$ for any $d \leq \log n$. Bollobás and Rosenfeld considered a relaxed sorting task, where the relative order of εn^2 pairs might still be unknown at the end. They showed that by performing $C_\varepsilon n^{3/2}$ comparisons, one can learn the order of $\binom{n}{2} - \varepsilon n^2$ pairs, where $\varepsilon \rightarrow 0$ as $C_\varepsilon \rightarrow \infty$. In contrast to the above existential bounds for 2-comparator based algorithms, our work provides *explicit* sorting algorithms. Our algorithms are efficient, they utilize t -comparators (allowing large values of t) and are asymptotically optimal, with respect to the above bounds.

Other related tasks were also considered in the literature. Alon and Azar [4] gave bounds on the number of comparisons required for approximate sorting and for selecting the median. Braverman, Mao, and Weinberg [10] considered the task of selecting the k -rank item, in a single round (and multiple rounds), and of partitioning an unordered array into the k -top and $(n - k)$ -bottom elements, in a single round. Their algorithms also work in the noisy-comparison setting, where each comparison is correct with probability $2/3$. Braverman, Mao, and Peres [9] extended the above results and gave an algorithm sorting the k -top elements in small number of rounds ($d = 1, 2$ and $d \geq 3$). They also give lower and upper bounds for this task, both in the noiseless and noisy-comparison setting.

A related approach for sorting is via *sorting networks* [7, 1, 25] and in particular, sorting networks of t -comparators, a task that was raised by Knuth [24, Question 54 in Section 5.3] and examined in [1, 30, 12, 15]. These are fixed networks of comparators with n inputs (each element is an input) and n outputs (the sorted elements). One main difference between our d -round sorting and a sorting network is that in the latter, each element appears exactly once as an input. Then, any comparator that gets this element as an input must appear in a different “round”. However, in a sorting algorithm, it is possible to give the same element to multiple comparators at the same round, and then form the total order out of the outcomes of all comparators.

Distributed sorting has appeared in the literature before, but it had a different meaning than the distributed sorting we consider here. Wegner [41] and Rotem, Santoro, and Sidney [34] considered the task of moving records around in a distributed network, so that they end up in a sorted manner (i.e., records that end up at the first site have keys which are strictly smaller than the records in the second site). These works mainly focused on the number of exchanged messages. We also briefly mention parallel VLSI sorting algorithms, e.g., [40, 36, 35, 23, 29]. Here the common setting is of $n \times n$ parallel processors, usually connected as a two-dimensional grid. Each processors holds one element at any given time and can transfer the element to a neighboring processors. The goal is that the elements will end up in an ordered alignment, i.e., the minimal element at the first processors, etc. This setting is somewhat similar to our case of $n = t^2$, if we think of a row or a column of processors as a single unit that can re-order the elements in that row or column according to their rank. Another sorting variant was considered by Patt-Shamir and Teplitsky [32] (building on [26]). Here, each computer starts with n records and needs to output their rankings in the global order of all n^2 records. Also unlike our task, each computer can sort any number of records that it holds (i.e., it is not limited to being a t -comparator).

As mentioned above, randomized *quicksort* with t -comparators was given by Beigel and Gill [8]. This algorithm features an optimal number of comparators, albeit it employs a large number of rounds, $d = O(\log_t n)$. A similar quicksort idea appeared earlier by Müller [28] for $t = \Omega(\log n)$, where the t -comparator is based on a systolic approach and takes $O(t)$ time to complete a single t -tuple sorting. Atallah, Frederickson, and Kosaraju [6] extended this result to the full range of t .

Mergesort with t -comparators is given in [37], and *cubesort* with t -comparators is presented in [14].

1.4 Organization

We formally state the problem of sorting with t -comparators, setting the relevant notations in Section 2. We discuss one-round deterministic sorting in Section 3. Our optimal 2-round randomized algorithm can be found in Section 4. The detailed analysis and missing proofs are deferred to the full version of this work. In Appendix A we provide some simulations

comparing our 2-round randomized algorithm with the state-of-the-art $O(\log_t n)$ -round t -quicksort algorithm, showing that the latter has in fact an expected number of rounds strictly larger than 4 when $n = t^2$.

1.5 Conclusions and Future Directions

In this work we studied the fundamental task of sorting n elements with t -comparators, where the sorting algorithm is limited to a small number of interactive rounds. This setting, while interesting on its own, fits in particular to distributed and parallel settings where interactive communication is very costly while computation resources are moderately costly.

We dealt with both deterministic and randomized algorithms. In the deterministic case, we established connections between optimal sorting algorithms in one round and combinatorial design theory. While this connection allows optimal sorting for certain values of n and t , it also suggests the impossibility for other values (e.g., $t = 6$). The question of the values of n, t for which optimal sorting exists is isomorphic to the long-standing combinatorial question of deciding the values of n, t for which the Steiner system $S(2, t, n)$ exists. We hope that an algorithmic approach could shed more light on this open question, e.g. through the construction of composition theorems similar to Lemma 15, or through explicit constructions for special cases.

Another interesting question is how the optimal number of t -comparators scales with the number of rounds. This topic was thoroughly examined in the literature for $t = 2$, and we extend the discussion to larger values of t . In the same vein, in the randomized setting, we design algorithms that use only two rounds but utilize the same asymptotic number of comparators as the optimal $O(\log_t n)$ -round t -quicksort algorithm.

We believe our findings might be useful in other distributed settings. For instance, in the Massively Parallel Computation model (MPC), where each worker machine performs the actions of one t -comparator, and all machines act in parallel. While our algorithm for $d = 1$ rounds requires a large number of machines (i.e., more than n/t), it might make sense to consider a larger amount of rounds and how it tradeoffs the number of machines in use. For instance, could a sublinear number of machines be sufficient for $d = O(1)$ rounds?

2 Preliminaries

Notations. For a positive integer n , we let $[n]$ denote the set $\{1, 2, \dots, n\}$. All logarithms are taken to base 2 unless otherwise noted. We say that an event happens *with high probability* in some parameter (usually, in the number of elements n), if the event occurs with probability at least $1 - 1/n^c$ for some positive constant $c \geq 1$.

Problem Statement. The elements are $A = \{a_1, a_2, \dots, a_n\}$. Each element has a value $val(a_i) \in [n]$. We assume that all values are unique, so that for any $i \neq j$, $val(i) \neq val(j)$, and all values in $[n]$ are covered.

A t -comparator is a device that gets t elements $\{a_{i_1}, \dots, a_{i_t}\}$ as an input, and outputs the respective order of their values. That is, it outputs a list j_1, \dots, j_t of indices, such that these are a permutation of i_1, \dots, i_t and it holds that $val(a_{j_1}) \leq val(a_{j_2}) \leq \dots \leq val(a_{j_t})$. Note that it is allowed to give as an input the same element multiple times (hence the inequality in the $val()$ values).

A *round of sorting* is any assignment of elements to (possibly multiple) comparators. The output of a single round of sorting is defined to be the output of all the comparators in that round, i.e., the relative order between any t elements compared by some comparator.

27:10 Sorting in One and Two Rounds Using t -Comparators

► **Definition 3.** Sorting n elements in d rounds via t -comparators is performing d rounds of sorting, where the assignment of round $i \leq d$ depends on the outputs of rounds $1, \dots, i - 1$. The assignment of elements to comparators is such that, for any possible assignment of values to the elements, there exists a single total ordering of the n elements that is consistent with all the outputs of the d rounds.

We will usually care about the number of t -comparators required to sort n elements. Let us denote $\text{OPT}(n, t, d)$ the minimal number of t -comparators required to sort n elements in d -rounds. In this paper we will focus on small values of d . In particular, in Section 3 we analyze the case of deterministic sorting in $d = 1$ rounds. In Section 4 we discuss randomized sorting with $d = 2$ rounds.

3 Sorting n elements in a single deterministic round

In this section we analyze sorting n elements with t -comparators in a single round. That is, we seek ways to assign elements to comparators that yield enough information to obtain a total-ordering of the elements. Since we restrict ourselves to a single round, we cannot adaptively select elements to compare based on previous result. Instead, all the assignments must be predetermined.

We begin with a few straightforward observations and facts. The following lemma is probably a well known folklore: if we are allowed to compare only pairs of elements ($t = 2$) and the comparisons are non-adaptive ($d = 1$), then *all* pairs of elements must be compared in order to obtain the total-ordering of the n element.

► **Lemma 4.** For $t = 2$, sorting n elements with 2-comparators in $d = 1$ rounds requires learning the relative order of each of the $\binom{n}{2} = \Theta(n^2)$ pairs of elements. Thus, $\text{OPT}(n, t = 2, d = 1) = \binom{n}{2}$.

Proof. Otherwise, there are two elements a_i, a_j that are not compared against each other. Let the two minimal elements (in the ranking) be a_i, a_j , respectively. Switching their relative order (i.e., letting the minimal elements be a_j, a_i , respectively) will not change the outputs of any of the comparators. Hence, there are two total ordering consistent with all the outputs, contradicting the fact that this is a sorting of n elements, Definition 3. ◀

► **Corollary 5.** $\text{OPT}(n, t, 1) \geq \binom{n}{2} / \binom{t}{2}$.

Proof. The proof of Lemma 4 extends to larger comparators. If two elements are not being compared by some comparator, let them be of minimal value and exchange their relative order to end up with two consistent total ordering. Thus, $\text{OPT}(n, t, 1)$ must provide enough comparators to compare all pairs.

Each t -comparator gives the ranking of t elements among themselves. That is, it allows us to learn the (pair-wise) order between at most $\binom{t}{2}$ pairs of elements. The statement immediately follows. ◀

Note that

$$\frac{\binom{n}{2}}{\binom{t}{2}} = \frac{n(n-1)}{t(t-1)} \geq \frac{n^2}{t^2} - \frac{n}{t^2}. \quad (1)$$

We can show that sorting with at most twice the amount of optimal comparators of Eq. (1) can be achieved for certain values of n, t ; sorting with at most three times the optimal is always possible.

► **Lemma 6.** *When $(t/2) \mid n$, $OPT(n, t, 1) < 2 \binom{n}{2} / \binom{t}{2}$. Otherwise, $OPT(n, t, 1) < 3 \binom{n}{2} / \binom{t}{2}$.*

Proof. Assume $(t/2) \mid n$. Split the n elements into $2n/t$ subsets of size $t/2$ each, $S_1, \dots, S_{2n/t}$. Now, for any $i, j \in [2n/t]$ compare the elements in $S_i \cup S_j$ using a t -comparator. It is immediate that any two elements will be compared in this process. The total number of comparators used is

$$\binom{2n/t}{2} = \frac{1}{2} \cdot \frac{2n}{t} \left(\frac{2n}{t} - 1 \right) = \frac{2n^2}{t^2} - \frac{n}{t}.$$

The above is clearly larger than twice Eq. (1), by noting that $n/t \geq 2n/t^2$ holds for $t \geq 2$.

However, when $t/2$ does not divide n , we need one additional subset $S_{2n/t+1}$ for the leftovers. This results with a total of $2n^2/t^2 + n/t$ comparators. When $t < \frac{-1+\sqrt{1+8n}}{2}$, this is still within a factor 2 of $\binom{n}{2} / \binom{t}{2}$. Otherwise, it is easy to see that we are within a factor 3 of the lower bound. Let us bound the ratio

$$\frac{\frac{2n^2}{t^2} + \frac{n}{t}}{\frac{n(n-1)}{t(t-1)}} = \frac{t-1}{t} \cdot \frac{2n+t}{n-1}.$$

The right hand side monotonically increases in t , and obtains its maximal value at $t = n - 1$. This yields

$$\frac{n-2}{n-1} \cdot \frac{3n-1}{n-1}.$$

This function monotonically increases in n (as can easily be seen from its derivation) and has a limit of 3 as $n \rightarrow \infty$. ◀

3.1 The case of a large t

Let us now give optimal sorting assignments with $d = 1$ for the case of a large comparator, $t = \Omega(n)$. To demonstrate the basic idea, assume $t = n - 1$. We argue that three comparators suffice in this case, which makes the bound in Lemma 6 tight for $n \geq 9$. First, we compare $\{a_1, \dots, a_{n-1}\}$ which gives a total-ordering for all elements but the last element, a_n , so we need to compare a_n with all the other elements. This can be done with by employing two additional comparators, e.g., comparing $\{a_n, a_2, \dots, a_{n-1}\}$ and $\{a_n, a_1, \dots, a_1\}$. Note that the second comparator is substantially under-utilized. This means that we could still perform sorting with only three comparators even for smaller values of t .

► **Lemma 7.** *For any $t \geq \frac{2}{3}n$, sorting n elements in a single round can be done with three comparators.*

Proof. The inputs to the three comparators are

$$\{a_1, \dots, a_t\}, \quad \{a_n, a_{n-1}, \dots, a_{t+1}, a_1, a_2, \dots, a_{\lceil t/2 \rceil}\}, \quad \text{and} \\ \{a_n, a_{n-1}, \dots, a_{t+1}, a_{\lceil t/2 \rceil+1}, \dots, a_t\}.$$

Note that any two elements a_i, a_j are being compared by some comparator, yielding all the information we need to obtain a single consistent total order of the elements.

Since $t \geq \frac{2}{3}n$, the second and third comparators get each $(n - (t + 1) + 1) + \lceil \frac{t}{2} \rceil \leq \lfloor \frac{3}{2}t \rfloor - t + \lceil \frac{t}{2} \rceil = t$ elements as input. Note that the ceiling/flooring matters only when t is odd. In this case $\frac{3}{2}t$ is fractional and since n must be an integer, we have $n \leq \lfloor \frac{3}{2}t \rfloor$. ◀

27:12 Sorting in One and Two Rounds Using t -Comparators

The above three comparators construction is tight, as it is impossible to sort n elements with only two comparators. The proof resembles the approach taken by Lemma 4 for the case of $t = 2$.

► **Lemma 8.** *For any $t < n$, sorting in one round cannot be achieved with two comparators.*

Proof. By a pigeonhole principle, there must exist (at least) two elements a_i, a_j that are not compared against each other. We make it so $\forall k \in [n] \setminus \{i, j\}, \text{val}(a_i) < \text{val}(a_k)$ and $\text{val}(a_j) < \text{val}(a_k)$. Then, it is impossible to determine which one of a_i, a_j is the minimal element. Specifically, setting $\text{val}(a_i) < \text{val}(a_j)$ gives the same comparator outputs as the case where $\text{val}(a_j) < \text{val}(a_i)$. This follows since they both are lower than any other element and no comparator has both of them as input. Then, there exists two total ordering consistent with the output of the comparators: one with $\text{val}(a_i) < \text{val}(a_j)$ and the other with $\text{val}(a_j) < \text{val}(a_i)$, contradicting Definition 3. ◀

3.2 Minimal sorting for a variety of parameters via design theory

Recall the proof of Corollary 5. It implies that every two elements must be compared against each other. This leads us to defining *minimal* sorting as follows.

► **Definition 9.** *Sorting is said to be minimal if equality holds in the equation in Corollary 5.*

That is, minimality is obtained when every two elements are compared against each other *exactly* once, and all the t -comparators are fully utilized. Then on the one hand there is no redundancy, and on the other hand all computational resources are fully used. Note that *optimality* means the minimal number of comparators needed to get all pairs compared against each other exactly once, but without requiring that all comparators are fully utilized.

While minimality implies optimality, the other direction does not hold. As demonstrated above for $2n/3 \leq t < n$, optimality is obtained with 3 comparators. However, minimality is not obtainable in this case. For instance, when $n = 10$, and $t = 7, 8, 9$ we have $\binom{n}{2} / \binom{t}{2} \in [1\frac{1}{4}, 2\frac{1}{7}]$, but, as we proved, exactly 3 comparators are necessary in all these cases, i.e., some comparator must be under-utilized regardless of the sorting algorithm.

► **Definition 10 (A Steiner System).** *A Steiner System with parameters $0 < c < t < n$, denoted $S(c, t, n)$, is a set \mathbb{P} of n elements (we will call points) and a set \mathbb{L} of objects (we will call lines), where each line is a subset of t points and it holds that any subset of c points is contained in exactly a single line.*

Corollary 5 and the discussion above imply the following.

► **Theorem 11.** *The Steiner system $S(2, t, n)$ is equivalent to a minimal sorting of n elements via t -comparators.*

Proof. Immediate from definitions. Every point is an element to sort, every line is a single comparator. Since any two points are contained exactly in a single line and since every line contains exactly t points, we obtain minimality. ◀

The above equivalence allows us to use known results about $S(2, t, n)$ to deduce cases for which minimal sorting is possible. The following is an immediate corollary of the known state-of-the-art about Steiner systems with $c = 2$, see e.g., [22, 19, 33].

► **Theorem 12.**

1. Let t be a power of a prime. Minimal sorting of $n = t^2$ elements is possible by employing $t^2 + t$ many t -comparators.
2. Let $t - 1$ be a power of a prime. Minimal sorting of $n = t^2 - t + 1$ elements is possible by employing $t^2 - t + 1$ many t -comparators.

Proof. (1) Follows from the fact that every field of size t implies a Steiner system $S(2, t, t^2)$ (an Affine Plane), see [31, Section 3.2]. (2) Follows from the fact that every field of size $t - 1$ implies a Steiner system $S(2, t, t^2 - t + 1)$ (a Projective Plane), see [31, Section 4.5]. We note that both constructions are explicit. ◀

The equivalence stated in Theorem 11 also yields some impossibilities on minimal sorting. It is well known that the Steiner system $S(2, 6, 36)$ does not exist. This problem, stated originally as a question about Latin Squares and known as the 36 officers problem, dates back to Euler [17] and was proven impossible by Terry [39]. Bruck and Ryster [11] extended this result and proved that Steiner systems of many other orders are also impossible.

► **Corollary 13** ([39, 11]). *Minimal sorting of $n = t^2$ elements (i.e., with exactly $t + t^2$ many t -comparators) is impossible for infinitely many values of t .*

Despite decades of research, a full characterization of values of t that admit a $S(2, t, t^2)$ system does not exist. In 1975, Willson [42] showed that for any t , a Steiner $S(2, t, n)$ system exists if and only if $t \mid n$ and $t(t - 1) \mid n(n - 1)$, *except for finitely many values of n* . This implies the following corollary

► **Corollary 14.** *For any t and large enough integer c , minimal sorting of $n = t^c$ elements is possible with $OPT(n, t, 1) = \binom{n}{2} / \binom{t}{2}$ many t -comparators.*

Indeed, for any $c \geq 1$ we have that $t \mid t^c$ and $(t - 1) \mid (t^c - 1)$ since $t^c - 1 = (t - 1)(t^{c-1} + t^{c-2} + \dots + 1)$. Our composition theorem, which is given in the next section (Lemma 15), gives explicit construction for some values of n, t . Finding explicit constructions for other values remains open.

3.3 A Composition Theorem

The above Theorem 12 applies only to the cases where $n = t^2$ or $n = t^2 - t + 1$ (for certain values of t). An interesting question is how to obtain a single-round sorting for other values of t and n , e.g., for $n = t^c$ elements, with $c \geq 3$. We partially answer this task by constructing a t^2 -comparator out of an optimal number of t -comparators. Operating recursively on larger n 's, this approach leads to the following theorem.

► **Lemma 15.** *Let t be power of a prime and let $n = t^{2^k}$ for some $k \in \mathbb{N}$. Then, minimal sorting of n elements with t -comparators is possible and employs $OPT(n = t^{2^k}, t, 1) = \binom{n}{2} / \binom{t}{2}$ many t -comparators.*

Proof. We prove that minimal sorting is possible by induction on k . The base case, $k = 1$ is given by Theorem 12(1).

For the induction step, assume we can sort $n' = t^{2^{k-1}}$ elements using $\binom{n'}{2} / \binom{t}{2}$ many t -comparators. We show how to sort $n = t^{2^k}$ elements with exactly $\binom{n}{2} / \binom{t}{2}$ t -comparators. Since n' is a power of a prime, Theorem 12 provides us a optimal (minimal) way to sort n elements using n' -comparators. Each n' -comparator can be implemented via an optimal

27:14 Sorting in One and Two Rounds Using t -Comparators

(minimal) number of t -comparators, by induction. The total number of t -comparator thus required to sort n elements is

$$\frac{\binom{n}{2}}{\binom{n'}{2}} \cdot \frac{\binom{n'}{2}}{\binom{t}{2}} = \frac{\binom{n}{2}}{\binom{t}{2}},$$

and this quantity is minimal by Corollary 5. ◀

As a corollary, the above composition theorem implies an explicit construction of a $S(2, t, t^{2^k})$ system for t a power of a prime and all integers $k > 0$.

4 Optimally sorting n elements in $d = 2$ randomized rounds

In Section 3, we studied optimal deterministic sorting in $d = 1$ rounds. We now wish to turn to the case of $d = 2$ rounds, trading-off one additional round for fewer comparisons. We study the *randomized* case since it allows us to reduce the number of comparisons considerably. Since for $d = 1$ we have already obtained an optimal deterministic solution, it makes sense to discuss randomized algorithms for $d > 1$. As randomized sorting with $O(\log n)$ comparators are well-known [21, 8], we wish to keep the number of rounds small, and focus on the case of $d = 2$. We design a fast randomized t -sorting algorithm, which is asymptotically optimal in the number of t -comparators used, restricted to algorithms with $d = 2$ rounds. In certain cases, for instance when $n = t^2$, the asymptotic number of t -comparators is optimal even without the round restriction. We discuss lower bounds on the number of t -comparators required for sorting in Section 4.1. In Section 4.2 we consider the special case of $d = 2$ and $n = t^2$ and in Section 4.3 we consider the more general case of $d = 2$ and arbitrary n and t . The detailed analysis is deferred to the full version. Our main result is Theorem 2, which we now recall.

► **Theorem 2** (main, randomized). *Let $t < n$ be given. There exists a (Las-Vegas) randomized sorting algorithm for n elements with two rounds, that utilizes $O\left(\max\left(\frac{n^{3/2}}{t^2}, \frac{n}{t}\right)\right)$ many t -comparators, with probability at least $1 - 1/n$.*

4.1 Lower bounds

Before describing our algorithms, let us recall the lower bound on the number of t -comparators, by Beigel and Gill [8].

► **Theorem 16** ([8]). *Sorting n elements requires utilizing at least $\frac{\log(n!)}{\log(t!)} = \frac{n \log n}{t \log t} (1 + o(1))$ many t -comparators.*

The proof stems from the fact that $\log(n!)$ bits of information are required to sort n elements, and that each comparator gives $\log(t!)$ bits of information. See Section II in [8].

The above lower bound allows any number of rounds. Alon and Azar [3] analyzed the average number of 2-comparators required to sort n elements in d rounds and proved the following.

► **Theorem 17** ([3]). *Sorting n elements in $d \leq \log n$ rounds, requires utilizing at least $\Omega(dn^{1+1/d})$ many 2-comparators on average.*

The above theorem could be used to derive lower bounds on sorting with t -comparators. Recall that each t -comparator compares at most $\binom{t}{2}$ pairs of elements. Then, the following lower bounds on the average number of t -comparators required in any randomized sorting is immediate.

► **Corollary 18.** *Sorting n elements in $d \leq \log n$ rounds, requires utilizing at least $\Omega(dn^{1+1/d}/t^2)$ many t -comparators on average.*

Because any average-case lower bound is also a worst-case lower-bound, if we plug in $d = 2$ in the above corollary, we obtain that our algorithm with $O(n^{3/2}/t^2)$ many t -comparator when $t < \sqrt{n}$, is asymptotically tight.

4.2 The simple special case of $n = t^2$

In this section we present Algorithm 2, which performs t -sorting of $n = t^2$ elements in two rounds and utilizes $O(t)$ many t -comparators. Note that by Theorem 16, this is asymptotically tight, even without the restriction to $d = 2$ rounds. Although our Algorithm 3 and Algorithm 4 described in Section 4.3 are strictly more general, as they apply to any n, t , for pedagogical reasons we first introduce the simplified and very natural Algorithm 2 that assumes the special case of $n = t^2$.

■ **Algorithm 2** A randomized 2-round sorting of $n = t^2$ elements with $O(t)$ many t -comparators.

Round 1:

- 1: Let P be a set of $t/2$ elements from A , each sampled uniformly and independently from A .
- 2: Partition A into subsets A_1, \dots, A_k of size $t/2$ each.
- 3: **for all** $i \in [k]$ **do**
- 4: Input $P \cup A_i$ into a comparator. ▷ k comparators
- 5: **end for**

Round 2:

- 6: Let $P = (p_1, \dots, p_{t/2})$ be the ordered elements in P . For $1 \leq i \leq t/2 - 1$, set S_i to contain all the elements which are greater than p_i but lower than p_{i+1} . Set S_0 to be all the elements lower than p_1 and $S_{t/2}$ be all the elements greater than $p_{t/2}$.
 - 7: **for all** $0 \leq i \leq t/2$ **do**
 - 8: Sort S_i via Lemma 6. ▷ at most $\sum_i 3|S_i|^2/t^2$ comparators
 - 9: **end for**
-

Recall our notations, where we wish to sort a set of $n = t^2$ elements, denoted $A = \{a_1, \dots, a_n\}$. We assume that t is even and that $(t/2) \mid n$, and set $k = n/(t/2)$. The algorithm works as follows. In the first round, we first sample $t/2$ elements uniformly from A . These will be our “pivots”. We then take the remaining elements of A and compare them to the pivots. That is, we split the remaining elements into $n/(t/2) - 1$ disjoint subsets of size $t/2$. We input each subset to a t -comparator together with (all) the $t/2$ pivots. After this step, for each element in A we know its relative position with respect to the pivots. Since we used the same pivots in each comparator, we can see the first round as the pivots splitting A into $t/2 + 1$ disjoint “buckets” such that all the elements in one bucket are strictly smaller (or strictly larger) than all elements in any other bucket. In the second round of the algorithm, we sort each bucket separately.

The first step utilizes $n/(t/2) = 2t$ comparators, one for each subset of A . In the second part, the number of comparators in use depends on the size of the buckets we need to sort, which is a random variable determined by the pivots we sample in the first round. In

expectation, each bucket is of size approximately⁴ $n/(t/2 + 1) \approx 2\frac{n}{t}$. If we assumed that the number of elements per bin is tightly concentrated around its mean, then we could deduce that sorting a single bucket using Lemma 6 would take $O(\frac{n^2}{t^4}) = O(1)$ comparators, and summing up over all $t/2 + 1$ buckets results in $O(\frac{n^2}{t^3}) = O(t)$ comparators overall, in expectation.

However, we cannot make such an assumption, since, while each bucket has $\approx 2\frac{n}{t}$ elements in expectation, there might be very large buckets, with, say, $O(\frac{n}{t} \log n)$ elements. Our analysis (which we perform only to the general case, in Section 4.3 below), is somewhat more intricate and shows that the event of a large bucket is rare enough so that amortizing across all the buckets, our algorithm still takes $O(t)$ comparators with high probability.

4.3 The general case: supporting any n, t

Algorithm 2 can be executed with any n, t . The problem is that this would come at a very high cost (measured in the number of t -comparators used). The main reason for this high cost is that Algorithm 2 has a tradeoff between the costs of the different rounds: the cost of the first rounds is $O(\frac{n}{t})$ and the cost of the second is $O(\frac{n^2}{t^3})$. While these two costs equal $O(t)$ for $n = t^2$, for arbitrary n and t these costs are no longer balanced and one of the rounds would have a relatively high cost. The idea behind Algorithm 3 depicted below,⁵ is to balance the costs of the phases, by carefully choosing the size of the pivot set and, as a result, the expected sizes of the buckets they yield.

■ **Algorithm 3** A randomized 2-round sorting for any n, t with $t \leq \sqrt{n}$.

Round 1:

- 1: Let P be a set of $m = \sqrt{n}$ elements from A , each sampled uniformly and independently from A .
- 2: Partition $A \setminus P$ into subsets A_1, \dots, A_k of size at most m each. $\triangleright k = \lceil (n - m)/m \rceil$
- 3: **for all** $i \in [k]$ **do**
- 4: Sort $P \cup A_i$ via Lemma 6.
- 5: **end for**

Round 2:

- 6: Let $P = (p_1, \dots, p_m)$ be the ordered elements in P . For $1 \leq i \leq m - 1$, set S_i to contain all the elements which are greater than p_i but lower than p_{i+1} . Set S_0 to be all the elements lower than p_1 and S_m be all the elements greater than p_m .
 - 7: **for all** $0 \leq i \leq m$ **do**
 - 8: Sort S_i via Lemma 6.
 - 9: **end for**
-

Assume that the first round randomly selects m pivots, which we denote by the set P . In order to “bucket” the n elements according to the pivots we need to compare them all with all the pivots. To that end, we split the set A into subsets A_1, \dots, A_k of size m (maybe except

⁴ To bound the expected size of each bucket, consider the sorted array of elements and uniformly select t pivots. Connect the beginning of the array to its end to form a cycle. Now consider all intervals between the pivots. The expected sum of the intervals, is roughly n . By linearity of expectation, we can consider disjoint “chunks” of intervals, each composed of t consecutive intervals. By symmetry, the expected lengths of all chunks are the same. Thus, each chunk must be, in expectation, about n/t elements long (ignoring constants).

⁵ Algorithm 3 is identical to Algorithm 1 described in the introduction and repeated here for convenience.

for the last subset), and compare each subset with the pivots. In contrary to Algorithm 2, we can no longer input $A_i \cup P$ into a t comparator. Instead, we need to implement a $2m$ -comparator out of t -comparators. We do so via Lemma 6, at the cost of $8m^2/t^2$ many t -comparators for a single simulated $2m$ -comparator.

Let us now analyze the expected cost of Algorithm 3. To calculate the cost of the first round, note that we now need $n/(2m)$ many (simulated) $2m$ -comparators each costing us $O(m^2/t^2)$ many t -comparators. Thus, the first round results in a total cost of $O(\frac{nm}{t^2})$. The expected cost of the second round is given as follows: since the set of pivots is sampled uniformly, the expected size of each bucket is $\approx 2n/t$ (Footnote 4). Oversimplifying again and assuming the number of elements per bin is tightly concentrated (which is not necessarily true for each bin), we get the following. By Lemma 6, each one of the $m+1$ buckets costs $O((n/m)^2/t^2)$ comparators in expectation. Overall, the expected cost in the second round is $O(\frac{n^2}{mt^2})$. Summing the costs of the two rounds, the expected cost of Algorithm 3 is $O(\frac{nm}{t^2} + \frac{n^2}{mt^2})$. Interestingly, this value is minimized when $m = \sqrt{n}$, irrespective of t . In the remainder, we simply set $m = \sqrt{n}$, and the cost becomes $O(\frac{n^{3/2}}{t^2})$.

The case of $t > \sqrt{n}$. The above analysis needs a little tweak to support the case of $t > \sqrt{n}$. In this case, the number of comparators-per-bucket given by the terms $O(m^2/t^2)$ and $O(n/m^2t^2)$ for the first and second round, respectively, is lower bounded by a single comparator, and thus should read $\max\{1, O(m^2/t^2)\}$ and $\max\{1, O(n/m^2t^2)\}$, respectively. Therefore, the choice of parameters needs to be adjusted. In the following we show a selection of parameters that optimize the case of $t > \sqrt{n}$, which yields Algorithm 4. We only give here a sketch of the (simplified) expected cost analysis, since the precise high-probability analysis see the full version.

■ **Algorithm 4** A randomized 2-round sorting for any n, t with $t > \sqrt{n}$.

Round 1:

- 1: $\tilde{m} = \lceil n/t \rceil$
- 2: Let P be a set of \tilde{m} elements from A , each sampled uniformly and independently from A .
- 3: Partition $A \setminus P$ into subsets A_1, \dots, A_k of size at most t each. $\triangleright k = \lceil (n - \tilde{m})/t \rceil$
- 4: **for all** $i \in [k]$ **do**
- 5: Sort $P \cup A_i$ via Lemma 6.
- 6: **end for**

Round 2:

- 7: Let $P = (p_1, \dots, p_{\tilde{m}})$ be the ordered elements in P . For $1 \leq i \leq \tilde{m} - 1$, set S_i to contain all the elements which are greater than p_i but lower than p_{i+1} . Set S_0 to be all the elements lower than p_1 and $S_{\tilde{m}}$ be all the elements greater than $p_{\tilde{m}}$.
- 8: **for all** $0 \leq i \leq \tilde{m}$ **do**
- 9: Sort S_i via Lemma 6.
- 10: **end for**

In Algorithm 4, We set the number of pivots to be $\tilde{m} = \lceil n/t \rceil$, and group the rest of the elements into subsets $\{A_i\}$ of size t each (instead of size \tilde{m}). We then continue with the sorting as before.

In the first round of the algorithm, we sort $k = O(n/t)$ sets, each of size $t + \tilde{m} = O(t)$. Thus, by Lemma 6 sorting each such bucket can be done using $c' = O(1)$ comparators resulting in $c'k = O(n/t)$ comparators in total. In the second round, each S_i has $O(n/\tilde{m}) = O(t)$ elements, in expectation. Assuming again our oversimplification that the number of

elements in each bin is tightly concentrated around its mean, we get by Lemma 6 that sorting each S_i takes $O(1)$ comparators. Since there are $\tilde{m} + 1$ such sets, the total number of comparators used in the second round is also bounded by $O(n/t)$.

In the full version we formally analyze the number of comparators used by these schemes (without the oversimplifying assumption) and show that it is concentrated around the stated value, i.e., we prove Theorem 2. As mentioned above, we only analyze Algorithm 3 since the analysis of Algorithm 4 is analogous. We stress again that the expected analysis presented above is oversimplified. Further, even with a simple and straightforward expected analysis, the dependencies of the events make it difficult to obtain high-probability concentration bounds, i.e., bounds that hold except with a polynomially small probability.

References

- 1 M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 1–9, 1983. doi:10.1145/800061.808726.
- 2 Selim G Akl. *Parallel sorting algorithms*, volume 12. Academic press, 1985.
- 3 N. Alon and Y. Azar. The average complexity of deterministic and randomized parallel comparison sorting algorithms. In *28th Annual Symposium on Foundations of Computer Science (SFCS 1987)*, pages 489–498, 1987. doi:10.1109/SFCS.1987.54.
- 4 Noga Alon and Yossi Azar. Sorting, approximate sorting, and searching in rounds. *SIAM Journal on Discrete Mathematics*, 1(3):269–280, 1988. doi:10.1137/0401028.
- 5 Noga Alon, Yossi Azar, and Uzi Vishkin. Tight complexity bounds for parallel comparison sorting. In *27th Annual Symposium on Foundations of Computer Science (SFCS 1986)*, pages 502–510, 1986. doi:10.1109/SFCS.1986.57.
- 6 Mikhail J. Atallah, Greg N. Frederickson, and S.Rao Kosaraju. Sorting with efficient use of special-purpose sorters. *Information Processing Letters*, 27(1):13–15, 1988. doi:10.1016/0020-0190(88)90075-0.
- 7 K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 307–314, 1968. doi:10.1145/1468075.1468121.
- 8 Richard Beigel and John Gill. Sorting n objects with a k -sorter. *IEEE Transactions on Computers*, 39(5):714–716, 1990. doi:10.1109/12.53587.
- 9 Mark Braverman, Jieming Mao, and Yuval Peres. Sorted top- k in rounds. In *Proceedings of the Thirty-Second Conference on Learning Theory*, volume 99 of *PMLR*, pages 342–382, 2019. URL: <https://proceedings.mlr.press/v99/braverman19a.html>.
- 10 Mark Braverman, Jieming Mao, and S. Matthew Weinberg. Parallel algorithms for select and partition with noisy comparisons. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 851–862, 2016. doi:10.1145/2897518.2897642.
- 11 Richard H. Bruck and Herbert J. Ryser. The nonexistence of certain finite projective planes. *Canadian Journal of Mathematics*, 1(1):88–93, 1949. doi:10.4153/CJM-1949-009-2.
- 12 YB Chiang. *Sorting networks using k -comparators*. PhD thesis, University of Cape Town, 2001. URL: <http://hdl.handle.net/11427/4871>.
- 13 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 4th edition, 2022.
- 14 Robert Cypher and Jorge L.C. Sanz. Cubesort: A parallel algorithm for sorting n data items with s -sorters. *Journal of Algorithms*, 13(2):211–234, 1992. doi:10.1016/0196-6774(92)90016-6.
- 15 Natalia Dobrokhotova-Maikova, Alexander Kozachinskiy, and Vladimir Podolskii. Constant-Depth Sorting Networks. In *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*, volume 251 of *LIPICs*, pages 43:1–43:19, 2023. doi:10.4230/LIPICs.ITCS.2023.43.

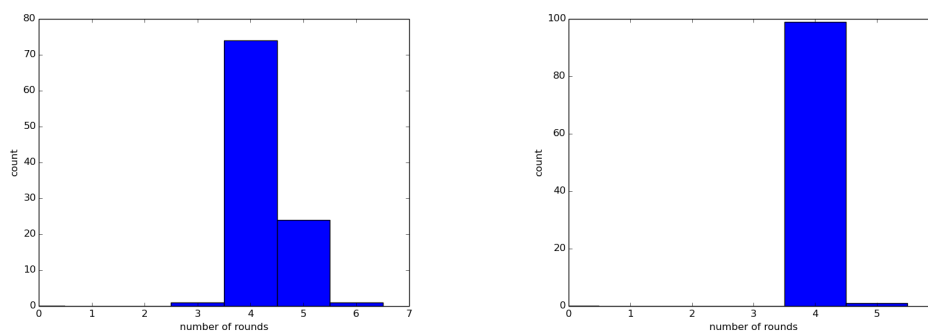
- 16 Vladimir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Comput. Surv.*, 24(4):441–476, December 1992. doi:10.1145/146370.146381.
- 17 Leonhard Euler. Recherches sur un nouvelle espèce de quarrés magiques. *Verhandelingen uitgegeven door het zeeuwisch Genootschap der Wetenschappen te Vlissingen*, pages 85–239, 1782.
- 18 Ran Gelles, Zvi Lotker, and Frederik Mallmann-Trenn. Sorting in one and two rounds using t-comparators. *CoRR*, abs/2405.12678, 2024. arXiv:2405.12678, doi:10.48550/arXiv.2405.12678.
- 19 Mike Grannell and Terry Griggs. An introduction to steiner systems. *Mathematical Spectrum*, 26(3):74–80, 1994.
- 20 Roland Häggkvist and Pavol Hell. Parallel sorting with constant time for comparisons. *SIAM Journal on Computing*, 10(3):465–472, 1981. doi:10.1137/0210034.
- 21 C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, January 1962. doi:10.1093/comjnl/5.1.10.
- 22 D. R. Hughes and F. Piper. *Design Theory*. Cambridge University Press, 1985.
- 23 Christos Kaklamanis and Danny Krizanc. Optimal sorting on mesh-connected processor arrays. In *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '92, pages 50–59, 1992. doi:10.1145/140901.140907.
- 24 Donald E. Knuth. *Art of computer programming, volume 3: Sorting and Searching*. Addison-Wesley Professional, 2nd edition, April 1998.
- 25 Tom Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C-34(4):344–354, 1985. doi:10.1109/TC.1985.5009385.
- 26 Christoph Lenzen and Roger Wattenhofer. Tight bounds for parallel randomized load balancing: extended abstract. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 11–20, 2011. doi:10.1145/1993636.1993639.
- 27 W. A. Martin. Sorting. *ACM Comput. Surv.*, 3(4):147–174, December 1971. doi:10.1145/356593.356594.
- 28 Heinrich Müller. Sorting numbers using limited systolic coprocessors. *Information Processing Letters*, 24(6):351–354, 1987. doi:10.1016/0020-0190(87)90109-8.
- 29 S. Olarin and S.Q. Zheng. Sorting n items using a p-sorter in optimal time. In *Proceedings of SPDP '96: 8th IEEE Symposium on Parallel and Distributed Processing*, pages 264–272, 1996. doi:10.1109/SPDP.1996.570343.
- 30 Bruce Parker and Ian Parberry. Constructing sorting networks from k-sorters. *Information Processing Letters*, 33(3):157–162, 1989. doi:10.1016/0020-0190(89)90196-8.
- 31 Abraham Pascoe. Affine and projective planes. Master's thesis, Missouri State University, 2018. MSU Graduate Theses. 3233. <https://bearworks.missouristate.edu/theses/3233>.
- 32 Boaz Patt-Shamir and Marat Teplitsky. The round complexity of distributed sorting: extended abstract. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '11, pages 249–256, 2011. doi:10.1145/1993806.1993851.
- 33 Colin Reid and Alex Rosa. Steiner systems $s(2, 4, v)$ -a survey. *The Electronic Journal of Combinatorics*, pages DS18–Feb, 2012.
- 34 Doron Rotem, Nicola Santoro, and Jeffrey B. Sidney. Distributed sorting. *IEEE Transactions on Computers*, C-34(4):372–376, 1985. doi:10.1109/TC.1985.5009389.
- 35 Isaac D. Scherson, Sandeep Sen, and Adi Shamir. Shear sort: a true two-dimensional sorting technique for VLSI networks. In *International Conference on Parallel Processing*, pages 903–908, 1986.
- 36 Claus-Peter Schnorr and Adi Shamir. An optimal sorting algorithm for mesh connected computers. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 255–263, 1986. doi:10.1145/12130.12156.
- 37 Feng Shi, Zhiyuan Yan, and Meghanad Wagh. An enhanced multiway sorting network based on n-sorters. In *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 60–64, 2014. doi:10.1109/GlobalSIP.2014.7032078.

- 38 Dhirendra Pratap Singh, Ishan Joshi, and Jaytrilok Choudhary. Survey of gpu based sorting algorithms. *International Journal of Parallel Programming*, 46:1017–1034, 2018. doi:10.1007/S10766-017-0502-5.
- 39 G. Tarry. Le problème de 36 officiers. *Compte Rendu de l'Association Française pour l'Avancement de Science Naturel*, 1900. vol. 1 (1900), 122-123; vol. 2 (1901), 170-203.
- 40 C. D. Thompson and H. T. Kung. Sorting on a mesh-connected parallel computer. *Commun. ACM*, 20(4):263–271, April 1977. doi:10.1145/359461.359481.
- 41 Lutz M. Wegner. Sorting a distributed file in a network. *Computer Networks (1976)*, 8(5):451–461, 1984. doi:10.1016/0376-5075(84)90007-2.
- 42 Richard M. Wilson. An existence theory for pairwise balanced designs, III: Proof of the existence conjectures. *Journal of Combinatorial Theory, Series A*, 18(1):71–79, 1975. doi:10.1016/0097-3165(75)90067-9.

APPENDIX

A Simulations: Our algorithm and the state-of-the-art algorithm

Let us compare our Algorithm 1 to the state-of-the-art quicksort algorithm with t -comparators, developed by Beigel and Gill [8]. Their algorithm works essentially as follows: randomly select $t/\log t$ pivot elements and use them to split all the elements into disjoint subsets. Now, recursively sort any subset of size exceeding t .



(a) $t = 10, n = 100$.

(b) $t = 100, n = 10000$.

■ **Figure 2** A histogram of the number of rounds required to the completion of the algorithm in [8] for the case of $n = t^2$ with (a) $t = 10$ and (b) $t = 100$. Each histogram is based on 100 repeated independent instances. In both t values, the average number of rounds is above 4.

The analysis in [8] proves that the number of t -comparators utilized throughout this algorithm is $\frac{n \log n}{t \log t} (1 + o(1))$, which is asymptotically optimal. The same analysis suggests the algorithm takes $\log_{m/2}(n)$ rounds, where $m = t/(2 \log(t) \ln(t))$. (The basis of the log in m is not defined in [8] and we take it to base e , yielding $m = t/2 \ln^2 t$.) It is easy to verify that this function approaches $\frac{\log n}{\log t}$ rounds, for sufficiently large t . In particular, for $n = t^c$, the function approaches c rounds as $t \rightarrow \infty$. We would like to compare this to our algorithm, that guarantees $d = 2$ rounds, regardless of t .

To be concrete, let us consider the case of $n = t^2$. In this case, $\log_{m/2}(t^2)$ tends asymptotically to 2 when $t \rightarrow \infty$. To demonstrate the behavior of the recursive algorithm we have performed Monte-Carlo simulations that measure the number of rounds it takes to sort $n = t^2$ elements, with $t = 10$ and $t = 100$. The results are depicted in Figure 2. Our findings indicate that, for these values of t , the average number of rounds for $n = t^2$ is not 2, but rather 4.