Sorting Under Forbidden Comparisons

Indranil Banerjee[1] and Dana Richards[2]

1   **Department Of Computer Science, George Mason University, Fairfax, USA**
      `ibanerje@gmu.edu`
2   **Department Of Computer Science, George Mason University, Fairfax, USA**
      `richards@cs.gmu.edu`

─── **Abstract** ───────────────────────────────────

In this paper we study the problem of sorting under forbidden comparisons where some pairs of elements may not be compared (forbidden pairs). Along with the set of elements V the input to our problem is a graph $G(V, E)$, whose edges represents the pairs that we can compare in constant time. Given a graph with n vertices and $m = \binom{n}{2} - q$ edges we propose the first non-trivial deterministic algorithm which makes $O((q+n)\log n)$ comparisons with a total complexity of $O(n^2 + q^{\omega/2})$, where ω is the exponent in the complexity of matrix multiplication. We also propose a simple randomized algorithm for the problem which makes $\widetilde{O}(n^2/\sqrt{q+n}+n\sqrt{q})$ probes with high probability. When the input graph is random we show that $\widetilde{O}(\min{(n^{3/2}, pn^2)})$ probes suffice, where p is the edge probability.

1   Introduction

Comparison based sorting algorithms is one of the most studied areas in theoretical computer science. The majority of the efforts have been focused on the uniform comparison cost model. Arbitrary non-uniform cost models can make trivial problems non-trivial, like finding the minimum [10, 16] . Thus it makes sense to consider a more structured cost. For example, a common cost model is the monotone[1] cost model. As shown in [16] the best one can do is to get an algorithm that is within a logarithmic factor of a cost optimal algorithm. However, the 1-∞ cost model in this paper is not monotonic. This model has comparison cost of 1 or ∞. A pair with cost ∞ is considered a "forbidden pair". The set of pairs with comparison cost 1, defines an undirected graph, $G(V, E)$, where V is the set of keys and E represents the allowed comparisons. We call G the comparison graph. Define E_f to be the set of forbidden pairs (edges). Let $|V| = n$ and $|E_f| = q$.

     Next we define the query model used in this paper. We don't get charged for checking whether an edge exists but are only charged for the comparisons made. The number of comparisons made or rather asked to the oracle is naturally defined as the *comparison complexity* or the *probe complexity*. No non-trivial ITB for the probe complexity is known in the standard decision tree model. We believe that the model is too weak for this purpose. For example, given a comparison graph G the number of different acyclic orientations of G gives an upper bound on the number of possible answers as each correspond to a unique

---

[1] By monotone we mean that the cost of comparing a pair is a monotone function of the values of the pair.

partial order. Since identifying G (up to isomorphism by verifying edges) is free and G has $\leq \prod_{v \in V}(d_v + 1) \leq n^n$[15] number of acyclic orientations we have the ITB of $\Omega(n \log n)$ for this problem. We believe this bound to be weak for this problem. The matter is further complicated if one is also given the guarantee that the graph G is *sortable*. We say G is sortable if G can be totally sorted. This restriction further reduces the number of possible answers for graphs with small number of edges. For example if G has $= n - 1$ edges then we can determine the unique total order by just making one comparison. Since any acyclic orientation of the edges of G must give a Hamiltonian path and G has $= n - 1$ edges, the edges must link consecutive vertices in the unknown order. A solitary probe is then used to determine the direction of this ordering. In this paper we take G to be arbitrary and not necessarily sortable. Hence by sorting G we mean determining the orientations of all the edges of G which may only get us a partial order on the vertices.

1.1 Prior Results

The problem of sorting with forbidden pairs is still open for the most part. It is closely related to the problem of partial sorting under a relation determining oracle. In this model we are given a set P of elements and a oracle $\mathcal{O}_r$ which is used to determine the relations between pairs of elements in P. The goal is to determine all the valid relations. Number of queries made to $\mathcal{O}_r$ is defined as the *query complexity*. Since there are $\Omega(2^{n^2/4})$[11] labeled posets with n elements, it immediately follows that the information theoretic bound (ITB) for the query complexity is $\Omega(n^2)$. This has been investigated for width bounded posets in [12], where the authors show that if P has width at most w (size of the largest anti-chain) then the ITB for the query complexity is $\Omega((w + \log n)n)$. They presented a query optimal algorithm for width bounded posets whose total complexity is $O(nw^2 \log \frac{n}{w})$. This algorithm can be generalized for any poset with an additional $\log w$ factor added to the the query complexity. Their results were the first major extension in this line of research after the seminal work by Faigle and Turán[14] which only showed the existence of such an algorithm. Another similar problem is the *local sorting* problem. In this problem V is an ordered set and for each $(u, v) \in E$ we want to determine their relative order. The problem is to determine if this can be done without sorting the entire set V, since the ITB for this problem is $\Omega(n \log (m + n)/n)$ in the standard comparison tree model (where m is the number of edges G). Currently no non-trivial deterministic algorithm is known for this problem. However, there is a randomized algorithm which makes optimal number of comparison with high probability [9]. Another related problem is the *partial order production problem*, where given a set T with an unknown total order we are interested in determining the partial order of another set S by comparing pairs in T. The goal is do this with minimum number of comparisons. The reader is referred to the survey by Cardinal et. al [8] which discusses some of these and other related problems in detail.

An example of a problem that uses the probe complexity model is the nuts and bolts problem. This is strictly not a sorting problem rather a matching one. In this problem one is given two sets of elements, a set of nuts and a set bolts. Elements in each set have distinct sizes and for each nut it is guaranteed that there exists a unique bolt of same size. Matching is performed by comparing a nut with a bolt. However, pairs of nuts or pairs of bolts cannot be compared. So in this case $G = K(N, B)$ is a complete bipartite graph with edges from the set of nuts N to the set of bolts B. This problem has been solved in the mid 1990s [3, 20]. The existence of a $O(n \log n)$ time deterministic algorithm was proved for it using the theory on bipartite expanders [3]. In the context of randomized algorithms, this problem has been studied in [17, 4]. The authors in [17] proposed a randomized algorithm that sorts G with

a probe complexity of $\widetilde{O}(n^{3/2})$ with high probability[2]. However their implementation uses as a sub-routine a poly-time uniform sampling algorithm to sample points from a convex polytope[13]. The authors did not discuss the exact bound on the total time complexity in their paper. At each step the algorithm either finds a *balancing edge*[3] or finds a subset of elements that can be sorted quickly. For an arbitrary G it is not guaranteed that a *balancing edge* always exists. However, when G is the complete graph there always exists a *balancing edge* that reduces the number of linear extension at-least by a factor of 8/11 [19].

1.2 Our Results

In this paper we propose the first non-trivial deterministic algorithm under the probe complexity model as well as a randomized algorithm. The results are expressed in terms of n and q. Expressing the results in terms of the number of forbidden edges fits naturally with the problem. First of all q and w are related, where w is the width of the poset P_G found after sorting G. We have $q \geq \#$ of incomparable pairs in $P_G \geq \binom{w}{2}$. Hence, $w = O(\sqrt{q})$. Although we cannot directly compare the probe complexity used in this paper with the query complexity in [12] it gives a better sense of the relatedness of the two models. Secondly, in the absence of any other structural properties of the input graph G, q gives a good indication of how difficult it is to sort G. For example, when $q = O(\log n)$, it is easy to see that one can sort in $O(n \log n)$ total time. To do this we pick an arbitrary pair of non-adjacent vertices and take out one of them, removing it from the graph. We do the same thing with the remaining graph until the graph remaining is a clique. It is clear that we had to take out at most $O(\log n)$ vertices. Then we sort this graph with $O(n \log n)$ comparisons and merge the vertices we had remove previously by probing all the remaining undirected edges, which is at most $O(n \log n)$. On the other extreme, if $|E| = \binom{n}{2} - q = O(n)$ then it can be shown that we need to make $\Omega(|E|)$ probes to determine the partial order, since the complete bipartite graph $K(A, B)$ with $|A| \ll |B|$ has many acyclic orientations [18, 15]. So in this case one has to probe most of the allowed edges.

The main contributions of this paper are as follows:

- Given a comparison graph G we propose a deterministic algorithm that sorts G with $O((q + n) \log n)$ probes. The total complexity of our algorithm is $O(n^2 + q^{\omega/2})$, where $\omega \in [2, 2.38]$ is the exponent in the complexity of matrix multiplication. We start by finding a set of large enough cliques in G and use its elements to determine a good pivot. This algorithm is applied recursively to induced subgraphs of G to generate a collection of partial orders. We then merge these partial orders in the final steps.
- We propose a randomized algorithm which sorts G with $O(n^2/\sqrt{n + q} + n\sqrt{q})$ probes with high probability. We use a random graph model for this purpose. The method uses only elementary techniques and unlike in Huang, et. al[17] has a total run time of $O(n^\omega)$ in the worst case.
- When G is a random graph with edge probability p we show that one can sort G with high probability using only $\widetilde{O}(\min(n^{3/2}, pn^2))$ probes.

The rest of this paper is organized as follows: in section 1.2 we introduce some definitions and lemmas. Section 2 details the proposed deterministic algorithm. In section 3 we introduce the randomized algorithm and its extension to random graphs.

---

[2] By *high probability* we mean that the probability tends to 1 as $n \to \infty$.
[3] An edge in G revealing whose orientation is guaranteed to reduce the number of linear extensions of the current partial order by a constant fraction. The pair of vertices incident to this edge is referred to as a *balancing pair*.

1.3   Definitions

Recall $G(V, E)$ is the input graph on the set V of elements to be sorted. A pair of vertices (u, v) can be compared if $(u, v) \in E$, otherwise, we say the pair is forbidden and is in E_f. The graph G is given to us by our adversary. Let G_i be the graph after i-edges have been oriented and P_i be the associated partial order. We denote the degree of a vertex v by $d(v)$ and $n(v) = n - 1 - d(v)$ is the number of vertices that are not adjacent to v. The set of neighbors of a vertex v is denoted by $N(v)$. We use the notation $E(A, B)$ we denote the set of edges between the sets of vertices $A, B \subset V$. We also define the little-o notation to remove any ambiguity from our exposition.

▶ **Definition 1.** If $f(n) \in o(g(n))$ then $f(n) \in O(g(n))$ but $f(n) \notin \Omega(g(n))$.

The following lemmas can be easily proven, hence we omit their proofs.

▶ **Lemma 2.** *Let* $\{f_1(n), f_2(n), ..., f_k(n)\}$ *be a finite set of non-negative monotonically increasing functions in n such that for $g(n)$:*
1. $\forall i \; f_i(n) \in o(g(n))$
2. $\sum_i f_i(n) \leq cg(n)$
If $F(n) = \sum_i f_i^2(n)$ *then* $F(n) \in o(g^2(n))$.

▶ **Lemma 3.** *Let* $T(n) = \sum_{i=1}^{k} T(n_i) + f(n)$ *where* $\sum_i n_i \leq \delta n$ *for some* $0 < \delta < 1$ *and* $f(n) \in o(n^2)$. *Then,* $T(n) \in o(n^2)$.

2   A Deterministic Algorithm For Restricted Sorting

First we look at a simple case where $q = O(n)$. We will use some of the main ideas from this algorithm to extend it to the general case. This initial algorithm will have a worse probe complexity than the main algorithm. In this algorithm we shall do case analysis based on whether a certain quantity is $o(n)$ or not. We acknowledge that this is not an algorithmic test. However, we use it in this algorithm to establish a framework for the second algorithm, which uses a traditional test and does not affect the claims made in this paper.

2.1   A Restricted Case

Assume $q \leq cn$ for some constant c. Let $R = \{v \in V \mid n(v) > c_1\}$ for some constant c_1, then $|R| \leq (2c/c_1)n$. This is obvious from the fact that $\sum_v n(v) \leq 2cn$. We choose $c_1 = 4c$. Let $S = V \setminus R$ and $G[S]$ be the induced subgraph generated by S. We have $|S| \geq n/2$ and if $v \in S$ then $n(v) \leq c_1$.

▶ **Claim 4.** *There exists a subset $X \subset S$ such that $|X| \geq \frac{n}{2(4c+1)}$ and $G[X]$ is a complete graph.*

Proof. We construct X explicitly. We start with $X = \{u\}$, where u is an arbitrary vertex in S. We pick successive vertices from S iteratively. Let v be last vertex to be added to X. Since v has at least $|S| - c_1$ neighbors, whenever we pick a neighbor of v from S to add to X we loose at most $c_1 + 1$ vertices (including the vertex we picked). Hence if we pick neighbors of v the size of X is at least $|S|/(c_1 + 1) \geq n/2(4c + 1)$.   ◀

Clearly the above procedure runs in $O(n^2)$ time and makes no comparisons. Now we are ready to describe our algorithm. The main algorithm is recursive and we have two levels of recursion. We shall break up the algorithm into several steps.

2.1.1 Initial Sorting

Given the input graph G, let X be a clique, with $|X| \geq n/2(4c + 1)$ (Claim 1). Let $Y = V \setminus X$. Note that $|Y| \leq n - n/2(4c + 1) = (8c + 1/8c + 2)n$. Now we sort X using $O(n \log n)$ comparisons as $G[X]$ is a complete graph. We can use a standard comparison based sorting algorithms for this purpose. Now we have two possibilities:

Case 1. If $|Y| = o(n)$, then we probe all edges of $G[Y]$ and $G[Y, X]$, where $G[Y, X]$ is the induced bipartite graph generated by the sets Y and X. Then we take the transitive closure of the resulting relations, which does not need any additional probes. It can be easily seen that the number of probe made in the previous step is $o(n^2)$. For the sake of contradiction if we assume that it is not so then $|X||Y| + |Y|^2/2 \geq dn^2$ for some d. Which implies $|Y| \geq dn$, since $|X| + |Y|/2 \leq n$. But then, $|Y| = \Omega(n)$, which is not true according to our earlier assumption. So, in this case we would have sorted V by making only $o(n^2)$ probes.

Case 2. Otherwise $|Y| \geq \delta n$, for some constant δ. In this case we recursively partition Y based on elements from X. We call this the partition step.

2.1.2 Partition Step

We will recursively partition both X and Y. To keep track of the current partition depth we rename X to X_{00} and Y to Y_{00}. We pick m_{00} the median of X_{00} (after X_{00} is sorted). Since $X_{00} \subset S$ we have $n(m_{00}) \leq c_1$. So m_{00} will be comparable to all but at most c_1 elements of Y_{00}. Let,

$$A_{00} = \{v \in Y_{00}| \ v \in N(m_{00})\}$$

and $B_{00} = Y_{00} \setminus A_{00}$. Note $|B_{00}| \leq c_1$. Now let U_{00} be the subset of A_{00} whose elements are $\geq m_{00}$ and the set L_{00} accounts for the rest of $A_{00} \setminus m_{00}$. Let X_{10} and X_{11} be the elements of X_{00} that are $<$ and $\geq$ to m_{00} respectively. We recursively partition the sets U_{00} and L_{00} using the medians of X_{10} and X_{11}. The B-sets are kept for later processing. We rename the sets U_{00} and L_{00} to Y_{10} and Y_{11}. So, the pairs (X_{10}, Y_{10}) and (X_{11}, Y_{11}) are processed as above generating the sets A_{10}, A_{11}, B_{10} and B_{11}. We continue doing this until the size of the X-set is $\leq c_2$, where c_2 is some constant. At this point we don't know the size of the Y-set paired with it. There are two cases we need to consider:

Case 1. $|Y| = o(n)$: Then we probe all the edges of $G[Y]$ and $G[X, Y]$ which uses at most $c_2|Y| + \binom{|Y|}{2}$ number of comparisons.

Case 2. $|Y| \geq \delta n$: Then we have $|Y| \geq \delta n$ for constant δ. Hence the graph $G[Y]$ can have at most $\leq (c/\delta)|Y|$ missing edges. This satisfies our initial premise that the number of missing edges in $G[Y]$ is linear in the number of vertices. Hence we can apply our initial strategy recursively[4]. That is we first find a large enough clique (which according to Claim 1 must exist) and then use it to partition the rest of the set Y.

Let us visualize using a partial recursion tree T (see Fig.1 below). We shall call T the partial recursion tree for reasons that will soon be clear. At the root we have the pair (X_{00}, Y_{00}). It has two children node (X_{10}, Y_{10}) and (X_{11}, Y_{11}) each having two children of

---

[4] Note that (c/δ) is an absolute constant. If the input graph has at most cn missing edges we apply the procedure recursively to subgraphs whose number of missing edges are at most (c/δ) times the number of vertices in the subgraph at any level of recusrion. This (c/δ) factor is not successively multiplied within each level of recursion.

their own and so on. Now at each level, the size of the X-set gets halved. So, the number of levels in T is at most $O(\log n)$. However, the Y-sets need not get divided with equal proportions. So, at the frontier (the deepest level) we will have nodes of the above two types, depending on the size of their corresponding Y-sets. Let the collection of these frontier nodes be partitioned in two sets Φ and Ψ corresponding to case 1 and case 2 respectively.

We can conclude that the total number of probes needed to compute all relations in Φ is $o(n^2)$. This follows from Lemma 1. Here we can map the size of the Y-sets of the nodes in the collection Φ to the functions $f_i(n)$. We know that the total elements in the union of these Y-sets is $\leq |Y_{00}| \leq (8c + 1/8c + 2)n$. The total number of probes will be $F(n)$ in worst case. What is the total number of probes on the internal nodes of T? We know that in the internal nodes we compare the median of the X-set with the elements of the A-set, which takes $|A|$ probes. Since the union of these A-sets cannot exceed the total number of vertices in $G(n)$, at each level of T we do at most $O(n)$ probes, totaling to $O(n \log n)$ probes over all the internal nodes.

Unlike the nodes in Φ, the nodes in Ψ recursively call the initial strategy using the input graph $G[Y]$. Let the probe complexity of our initial strategy be $Q(n)$. Then the recursion for Q is as follows:

$$Q(n) = \sum_{i=1}^{|\Psi|} Q(n_i) + o(n^2)$$

Here we assume that the nodes in Ψ are indexed according to some arbitrary order. We can solve this recurrence using Lemma 2 giving $Q(n) \in o(n^2)$, since $\sum_{i=1}^{|\Psi|} n_i \leq (8c + 1/8c + 2)n$. Note here that $|\Psi|$ is bounded by a constant since the size of the Y-sets are $\Omega(n)$.

We call $\hat{T}$ the full tree. All leaf nodes in $\hat{T}$ are in Φ. It is straightforward to show that $\hat{T}$ has $O(\log^2 n)$ levels. Since any of the leaf nodes of T has $|Y| \leq \beta n$ (where $\beta = (8c+1/8c+2)$), its subtree in $\hat{T}$ can have at most $\alpha \log \beta n = \alpha \log n - \alpha \beta$ levels, and any of its leaves having at most $\alpha \log n - 2\alpha\beta$ levels and so on for some constant α.

2.1.3 Merge Step

Once we have completed building $\hat{T}$ we proceed with the final stage of our algorithm. Recall that during the forward partition step we had generated many of these B-sets in the internal nodes of $\hat{T}$. Now we start from the leaves of $\hat{T}$ and proceed upwards. Each pair of leaf nodes l, r sharing a common parent p, sends a partial order to it (computed as in case 1). When we merge this two partial orders we know that no extra probes are needed since they have already been split by the median of the X-set of p. What remains is to probe all edges between the B-set in p and elements in this partial order (which constitutes the set of elements $A \cup X$ of the node p) as well as the edges in $G[B]$. Then we pass the resulting partial order to the parent of p, and so on. Since the size of the B-sets are bounded by c_1 (at any level in $\hat{T}$), total number of probes we make is then $\leq c_1 \sum_i (|A_i| + |X_i| + c_1)$. The sum is taken over all the nodes in that level. Hence this is bounded by $c_1 n$, so at each level we do at most $O(n)$ probes in the backward merging step. Since there are at most $O(\log^2 n)$ levels, it totals to $O(n \log^2 n)$ additional probes. Adding this to the probe cost of partitioning in the forward step does not effect the total probe complexity, which was $o(n^2)$. The final step is to compute the transitive closure of the resulting set of relations, which can be done without any additional probing. Since computing the transitive closure is equivalent to boolean matrix multiplication[21] the total complexity is $O(n^\omega)$.
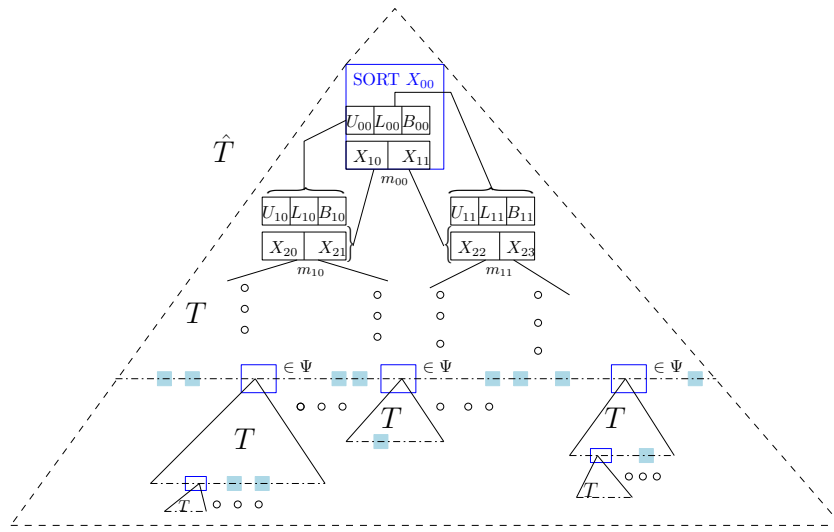
Figure 1 Visualizing the steps. At the bottom of T the shaded boxes represents the Φ-nodes and the blue rectangles the Ψ-nodes. The outer dashed triangle represents the full tree $\hat{T}$. The tree $\hat{T}$ is created during the partitioning step and in the merge step we start from the deepest leaves of $\hat{T}$ and move upwards.

2.2 The General Case

We will define the sets R and S analogously to section 2.1. We have $R = \{v \in V \mid n(v) > c_1 q/n\}$ for some constant c_1. With $c_1 = 4$, we get $|R| \leq \delta_1 n$ where $\delta_1 \leq 2/c_1 = 1/2$. Hence $|S| \geq (1 - \delta_1)n \geq n/2$. Now we will apply Claim 1 successively to construct a "big-enough" set $X \subset S$ which we will use to find an approximate median of V. This set X consists of disjoint subsets X_i such that $G[X_i]$ is a clique.

2.2.1 Constructing X

Let us define $S_i = S \setminus \bigcup_{j=1}^{i} X_j$. We construct the first clique $X_1 \subset S$ using the method detailed in Claim 1. There are two cases:

Case 1. $q < n$: In this case we can show that $|X_1| \geq (n/2)/(c_1 q/n + 1) \geq n/10$. We take the first $n/10$ elements and keep the rest for the second round. Now we construct the second clique X_2 from S_1 which has at least $2n/25$ vertices. We let $X = X_1 \cup X_2$. Hence X has at least $9n/50$ vertices.

Case 2. $q \geq n$: In this case we have $|X_1| \geq (n/2)/(c_1 q/n + 1) \geq n^2/10q$. Again we take $|X_1| = (1/10)n^2/q$ discarding some vertices if necessary. Similarly we construct $X_2 \subset S_1$. It can be shown that $|X_2| \geq (n^2/10q)(1 - n/5q)$ and we keep $(n^2/10q)(1 - n/5q)$ vertices in X_2 and the rest are discarded to be processed the next round. In general for the clique X_r we have $|X_r| \geq (n^2/10q)(1 - n/5q)^{r-1}$. Now we let $X = \bigcup_{i=1}^{r} X_i$. We will show that $|X| \geq \delta_2 n$ for some constant $\delta_2 > 0$. We let $r = 5q/n + 1$. Then we have

$$|X_r| \geq (n^2/10q)(1 - n/5q)^{r-1} \geq (n^2/10q)(1 - n/5q)^{5q/n} > 3n^2/100q$$

since $q \geq n$. Hence, $|X| = \sum_{i=1}^{r} |X_i| \geq r|X_r| \geq (9/50)n$, giving $\delta_2 = 9/50$. Now for each X_i ($1 \leq i \leq r$) we keep a subset Y_i of size $|X_r|$ and throw away the rest. Clearly, for each i, the induced sub-graph $G[Y_i]$ is also a clique. Let $Y = \bigcup_{i=1}^{r} Y_i$. We also have $|Y| \geq (9/50)n$.

2.2.2 Computing An Approximate Median Of V

We shall compute an approximate median with respect to all the vertices (the set V) and not just the set S. We will find a median element that divides the set V in constant proportions. This can be done easily using the set Y. For each Y_i we find its median using $\Theta(|Y_i|)$ probes since $G[Y_i]$ is a complete graph. Let this median be m_i and M be the set of these r medians. Since $m_i \in S$, $n(m_i) \leq 4q/n$. We define the upper set of $m \in M$ with respect to a set $A \subset V$ (m may not be a member of A) as $U(m, A) = \{a \in A \mid a > m\}$. Similarly we define the lower set $L(m, A)$. We want to compute the sets $U(m, Y)$ and $L(m, Y)$. However, m may not be neighbors of all the elements in Y. So we compute approximate upper and lower sets by probing all the edges in $E(\{m\}, Y \setminus \{m\})$. These sets are denoted by $\widetilde{U}(m, Y)$ and $\widetilde{L}(m, Y)$ respectively. It is easy to see that there exists some $m \in M$ which divides Y into sets of roughly equal sizes (their sizes are a constant factor of each other). In fact the median of M is such an element. However the elements in M may not all be neighbors of each other hence we will approximate m using the ranks of the elements in M with respect to the set Y (which is $|\widetilde{L}(m, Y)|$). Next we prove that the element m^* is an approximate median of M, picked using the above procedure, is also an approximate median of Y.

▶ **Claim 5.** *The element m^* picked as described above is an approximate median of Y.*

Proof. First we show that the median of M is an approximate median of Y. This can be easily verified. Let us take the elements in M in sorted order $(m_1, ..., m_r)$, so the median of M is $m_{\lfloor r/2 \rfloor}$. Now $L(m_{\lfloor r/2 \rfloor}, Y) \geq \bigcup_{i=1}^{\lfloor r/2 \rfloor} L(m_i, Y_i)$. Since, the sets Y_i are disjoint and $L(m_i, Y_i) \geq |X_r|/2$, we have $|L(m_{\lfloor r/2 \rfloor}, Y)| \geq |X_r|r/4$ (ignoring the floor). Similarly we can show that $|U(m_{\lfloor r/2 \rfloor}, Y)| \geq |X_r|r/4$. Hence $m_{\lfloor r/2 \rfloor}$ is an approximate median of Y. Now we show that $|\,|L(m^*, Y)| - |L(m_{\lfloor r/2 \rfloor}, Y)|\,| < 4q/n$. Consider the sorted order of elements in M according to $|\widetilde{L}(m^*, Y)|$. Since each element in $m \in M$ has at most $4q/n$ missing neighbors in Y, we have $|\,|\widetilde{L}(m, Y)| - |L(m, Y)|\,| < 4q/n$. So the rank of an element in the sorted order is at most $4q/n$ less than its actual rank. Thus an element m^* picked as the median of M using its approximate rank $|\widetilde{L}(m, Y)|$ cannot be more than $4q/n$ apart from $m_{\lfloor r/2 \rfloor}$ in the sorted order of Y. Hence

$$|L(m^*, Y)| \geq |X_r|r/4 - 4q/n \geq 9n/200 - 4q/n \geq n/40 \tag{1}$$

whenever $n^2 \geq 200q$. In an identical manner we can show that $|U(m^*, Y)| \geq n/40$. Hence, m^* is an approximate median of Y. When $q < n$ we just take m^* as the median with the higher $|\widetilde{L}(\cdot, Y)|$ value, which guarantees $|L(m^*, Y)| \geq n/40$ whenever $n^2 \geq 800q/13$. So we take $n^2 \geq 200q$ to cover both the cases. ◀

It immediately follows that m^* is also an approximate median of V with both $|L(m^*, V)|$ and $|U(m^*, V)|$ lower bounded by $n/40$. Lastly, we note that the above process of computing an approximate median makes $\Theta(q + n)$ probes. This follows from the fact that computing the medians makes $\Theta(n)$ probes in total and for each of the $\leq 5q/n + 1$ medians we make $O(n)$ probes.

2.2.3 A Divide-And-Conquer Approach

Now that we have computed an approximate median of V we proceed with an recursive approach. Let m^* be the median. As in section 2.1 we partition V into three sets U, L and B. The U and L are the upper and lower sets with respect to m^*. B is the set of vertices that do not fall into either, that is, they are non-neighbors of m^*. Since $m^* \in S$ we have $|B| \leq 4q/n$.

We recursively proceed to partially sort the sets U and L with the corresponding graphs $G[U]$ and $G[L]$ and keep B for later processing (as we did in the merging step previously). Like before we can imagine a recursion tree T. Let E_{f_P} be the set forbidden edges in $G[P]$. We take $n_P = |P|$ and $q_P = |E_{f_P}|$. For each node $P \in T$ there are two cases:

Case 1. When $n_P^2 \geq 200q_P$ we recursively sort P. In this case we can guarantee that the approximate median m_P^* of P will satisfy equation (1). That is both $|L(m_P^*, P)|$ and $|U(m_P^*, P)|$ is $\geq n_P/40$.

Case 2. Otherwise we probe all edges in $G[P]$. In this case P will become a leaf node in T. It can be easily seen that the depth of the recursion tree is bounded by $O(\log n)$ since at each internal node P of T we pass sets of constant proportions (where the size of the larger of the two set is upper bounded by $(39/40)n_P$) to its children nodes.

2.2.4 Merge Step

In this step we start with the leaves of T and proceed upwards. A parent node P gets two partial orders from its left and right children respectively. Then it probes all the edges between its B-set and these partial orders to generate a new partial order and pass it on to its own parent. This step works exactly as the "merge step" of the previous algorithm. Only difference is that the B-sets here may not be of constant size but of size $\leq 4q/n$.

2.2.5 Probe Complexity

We can determine the probe complexity by looking at the recursion tree T. First we compute it for the forward partition step. At each internal node of T we compute a set of medians and pick one element from it appropriately chosen. Then we partition the set of elements at the node by probing all edges between the selected element and rest of the elements in the node. As mentioned before this only takes $\Theta(q_P + n_P)$ probes for some internal node P. We assume that all the leaves of T are at the same depth, otherwise we can insert internal dummy nodes and make it so. At each level of T the sum total of all the vertices in every node is $\leq n$ and the sum total of the forbidden edges is $\leq q$. Hence we do $O(q + n)$ probes at any internal level of T. So for a total of $O(\log n)$ internal levels in T the number of probes done is $((q + n) \log n)$ in the forward partition step. If P is a leaf node then we probe all edges in $G[P]$. There are at most $\binom{n_P}{2} - q_P$ edges in $G[P]$. Since P is a leaf node, according equation 1, $n_P^2 < 200q_P$. Hence we make $\binom{n_P}{2} - q_P = O(q_P)$ probes. Summing this over all the leaves gives a total of $O(q)$ probes. Hence the total probe complexity during the forward step is $O((q + n) \log n)$.

Now we look at the merging step. Merging happens only at the internal nodes. Lets look at an arbitrary internal level of T. At each node P of this level we probe all the edges in $E(B_P, U_P \cup L_P \cup m_P^*)$ and in $G[B_P]$. Note that we do not have to make any probes between U and L as they were already separated by the approximate median m_P^*. Hence the total number of probes made in this node is $\leq (|U_P| + |L_P| + |B_P| + 1)|B_P| \leq (n_P)(4q_P/n_P) \leq 4q_P$. Summing over all the nodes at any given level gives us $O(q)$ as the probe complexity per level. So the total probe complexity in the merging stage is $O(q \log n)$. Hence, combining the probes made during the partition step and the merge step we see that the total probes needed to sort V is $O((q + n) \log n)$.

2.2.6 Total Complexity

Now we look at the total complexity of the previous procedure. Again the analysis is divided into forward step and the merge step. In the forward step at each node P we perform

$O(n_P^2)$ operations. This includes computing the degrees, finding the cliques, computing the approximate median. So at any level of T, regardless of it being an internal level or not, we perform $O(n^2)$ operations. Hence it totals to $O(n^2 \log n)$ operations in the forward step. However this is a conservative estimate and we can remove the $\log n$ factor as argued below: we can define the recurrence for the forward computation as,

$$T(n) = \begin{cases} T(n/40) + T(39n/40) + O(n^2) & n^2 \geq 200q \\ O(q) \text{ Otherwise} \end{cases} \tag{2}$$

This follows from the previous discussion. If we don't recurse on a node we guarantee that $n_P^2 < 200q_p$ for that node. Hence, we have $T(n) = O(n^2 + q)$ using the Akra-Bazzi method[2]. In the merge step, we only make $O(q_P)$ comparisons at any given node. We compute transitive closures only at the leaves. However for any leaf P we have $n_P^2 < 200q_P$. Hence computing the transitive closure of $G[P]$ takes $O(q_P^{\omega/2})$ time. Hence, the total complexity of the above procedure is $O(n^2 + q^{\omega/2})$. We summarize the results in this section with the following theorem:

▶ **Theorem 6.** *Given a graph $G(V, E)$ of n vertices having q forbidden edges, one can compute the partial order of V with $O((q + n) \log n)$ comparisons and in total $O(n^2 + q^{\omega/2})$ time.*

Proof. Follows from the discussions in this section. ◀

3 A Randomized Algorithm

In this section we look at a more direct way of sorting by making random probes. The proposed method is inspired by the literature on two-step oblivious parallel sorting [1, 7] algorithms, in particular on a series of studies by Bollobás and Brightwell showing certain sparse graphs can be used to construct efficient sorting networks [6, 5]. It was shown that if a graph satisfies certain properties then probing its edges and taking the transitive closure of the results would yield large number of relations. Then we just probe the remaining edges that are not oriented, which is guaranteed (with high probability) to be a "small" set.

The main idea is as follows: Let $\mathcal{H}_n$ be a collection of undirected graphs on n vertices having certain properties. A transitive orientation of a graph $H(V, E) \in \mathcal{H}_n$ is an ordering of V and the induced orientation of the edges of H based on that ordering. Let σ be an ordering on V and $P(H, \sigma)$ be the partial order generated by this ordering σ on H. It is a partial order since H may not be sortable. Let $\mathcal{P} = P(H, \sigma)$ and $t(P)$ be the number of incomparable pairs in $\mathcal{P}$. We want H to be such that $t(p)$ is small. If that is the case then $\mathcal{P}$ will have many relations and if H is sparse then we can probe all the edges of H and afterwards we will be left with probing only a small number of pairs. These are pairs which were not oriented during the first round of probing and after the transitive closure computation. A graph H is *useful* to our purpose if every transitive orientation of H results in many relations. We want to find a collection $\mathcal{H}_n$ such that every graph in it is useful with high probability.

We extend the results in [6, 5] to show that a collection of certain conditional random graphs are useful, with high probability. In our case this random graph will be a spanning subgraph of the input graph G. Here we recall an important result from [6] (Theorem 7) which we will use in our proof.

▶ **Theorem 7** ([6]). *If G is any graph on n vertices and G satisfies the following property:*
Q1 *Any two subsets A, B of vertices having size l have at least one edge between them.*
Then, the number of incomparable pairs in $P(G, \sigma)$ is at most $O(nl \log l)$ for any σ.

The input graph G is chosen by our adversary. However, we show that any random spanning subgraph of G with an appropriate edge probability will satisfy Q1 with high probability. Let $H_{n,p}(G)$ be a random spanning subgraph of G, where $H_{n,p}(G)$ has the same vertex set as G and a pair of vertices in $H_{n,p}(G)$ has an edge between them with probability p if they are adjacent in G, otherwise they are also non-adjacent in $H_{n,p}(G)$. All we need to prove is that any random spanning subgraph $H_{n,p}(G)$ given G with n-vertices and edge probability p will satisfy Q1 with high probability. Since G has at most q forbidden edges any two subsets of vertices A, B (not necessarily distinct) of size l must have at least $\binom{l}{2} - q$ edge between them. Let E_{AB} be the event that the pair (A, B) is bad (they have no edges between them), then the probability $S_{n,p}$ that there exists a bad pair is:

$$S_{n,p} := \mathbb{P}(\sum_{i,j} E_{A_i B_j}) \leq \sum_{i,j} \mathbb{P}(E_{A_i B_j}) \leq \sum_{i,j} (1-p)^{e(A_i, B_j)} \tag{3}$$

where the sum is taken over all such $\binom{n}{l}^2$ pairs of subsets, and the number of edges between the two sets A and B in G is $e(A, B) \geq \binom{l}{2} - q$. So we have,

$$S_{n,p} \leq \binom{n}{l}^2 (1-p)^{\binom{l}{2}-q} \leq \binom{n}{l}^2 e^{-p(\binom{l}{2}-q)} \quad \text{Since, } e^{-x} \geq 1-x$$

$$\leq \left(\frac{en}{l}\right)^{2l} e^{-p(\binom{l}{2}-q)} \leq \exp(2l(\log en/l) - p(\binom{l}{2} - q))$$

Hence $S_{n,p} \to 0$ as $n \to \infty$ whenever $\exp(2l(\log en/l) - p(\binom{l}{2} - q)) = o(1)$. Given $q < \binom{n}{2}$ it is always possible to find appropriate values for p and l as functions of q and n such that $S_{n,p} = o(1)$. Given some value for the pair (p, l), we see that in the first round we make $O(pn^2)$ probes with high probability and in the second round $O(nl \log l)$ probes (for the remaining unoriented edges) again with high probability. So the total probe complexity is $\widetilde{O}(pn^2 + nl)$. With some further algebra it can be shown that this is $\widetilde{O}(n^2/\sqrt{q+n} + n\sqrt{q})$. We summarize this section with the following theorem:

▶ **Theorem 8.** *Given a graph G on n vertices and q forbidden edges one can determine the partial order on G with high probability in two steps by probing only $\widetilde{O}(n^2/\sqrt{q+n} + n\sqrt{q})$ edges in total and in $O(n^\omega)$ time.*

Proof. Follows from the preceding discussions. ◀

3.1 When G Is A Random Graph

The above technique can easily be extended for the case when the input graph is random. Let $G_{n,p}$ be the input graph having n-vertices and an uniform edge probability p. For such a graph we can use equation (3) to bound $S_{n,p}$ as follows:

$$S_{n,p} \leq \binom{n}{l}^2 (1-p)^{l^2} \leq \exp(-pl^2 + 2l \log n)$$

Hence, we can choose any $l > 2 \log n/p$ such that $S_{n,p} \to 0$ as $n \to \infty$. Let $l = 3 \log n/p$. Using Theorem 2 we have $t(G_{n,p}) = \widetilde{O}(nl) = \widetilde{O}(n/p)$. Since $G_{n,p}$ has $pn^2/2$ edges (with

high probability) the critical value of p when $t(G_{n,p}) = pn^2/2$ is $\widetilde{O}(1/\sqrt{n})$. Let this be $\hat{p}$. Hence if $p > \hat{p}$, we can sort by making only $\widetilde{O}(n^{3/2})$ comparisons. Since given $G_{n,p}$ we can construct an induced subgraph $G_{n,\hat{p}}$ and use it as the random graph in our previous construction. Otherwise we just probe all the edges which makes $O(pn^2)$ comparisons. Thus we can sort $G_{n,p}$ with at most $\widetilde{O}(\min(n^{3/2}, pn^2))$ comparisons with high probability. Hence, we get an elementary technique to sort a random graph with at most $\widetilde{O}(n^{3/2})$ comparisons. The algorithm in [17] has a slightly better bound of $\widetilde{O}(n^{7/5})$ comparisons. However, the total runtime of the algorithm in [17] is only polynomially bounded when p is small. In our algorithm we need compute the transitive closure only twice making it run in $O(n^\omega)$ total time.

References

1 Miklós Ajtai, János Komlós, William Steiger, and Endre Szemerédi. Almost sorting in one round. *Randomness and Computation*, 5:117–125, 1989.

2 Mohamad Akra and Louay Bazzi. On the solution of linear recurrence equations. *Comp. Opt. and Appl.*, 10(2):195–210, 1998. URL: `http://dx.doi.org/10.1023/A:1018373005182`, `doi:10.1023/A:1018373005182`.

3 Noga Alon, Manuel Blum, Amos Fiat, Sampath Kannan, Moni Naor, and Rafail Ostrovsky. Matching nuts and bolts. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia.*, pages 690–696, 1994. URL: `http://dl.acm.org/citation.cfm?id=314464.314673`.

4 Stanislav Angelov, Keshav Kunal, and Andrew McGregor. Sorting and selection with random costs. In *LATIN 2008: Theoretical Informatics, 8th Latin American Symposium, Búzios, Brazil, April 7-11, 2008, Proceedings*, pages 48–59, 2008. URL: `http://dx.doi.org/10.1007/978-3-540-78773-0_5`, `doi:10.1007/978-3-540-78773-0_5`.

5 Béla Bollobás and Graham Brightwell. Graphs whose every transitive orientation contains almost every relation. *Israel Journal of Mathematics*, 59(1):112–128, 1987.

6 Béla Bollobás and Graham R. Brightwell. Transitive orientations of graphs. *SIAM J. Comput.*, 17(6):1119–1133, 1988. URL: `http://dx.doi.org/10.1137/0217072`, `doi:10.1137/0217072`.

7 Béla Bollobás and Moshe Rosenfeld. Sorting in one round. *Israel Journal of Mathematics*, 38(1-2):154–160, 1981.

8 Jean Cardinal and Samuel Fiorini. On generalized comparison-based sorting problems. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 164–175, 2013. URL: `http://dx.doi.org/10.1007/978-3-642-40273-9_12`, `doi:10.1007/978-3-642-40273-9_12`.

9 Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M. Jungers, and J. Ian Munro. An efficient algorithm for partial order production. *SIAM J. Comput.*, 39(7):2927–2940, 2010. URL: `http://dx.doi.org/10.1137/090759860`, `doi:10.1137/090759860`.

10 Moses Charikar, Ronald Fagin, Venkatesan Guruswami, Jon M. Kleinberg, Prabhakar Raghavan, and Amit Sahai. Query strategies for priced information. *J. Comput. Syst. Sci.*, 64(4):785–819, 2002. URL: `http://dx.doi.org/10.1006/jcss.2002.1828`, `doi:10.1006/jcss.2002.1828`.

11 SD Chatterji. The number of topologies on n points, kent state university. *NASA Technical Report*, 1966.

12 Constantinos Daskalakis, Richard M. Karp, Elchanan Mossel, Samantha Riesenfeld, and Elad Verbin. Sorting and selection in posets. *SIAM J. Comput.*, 40(3):597–622, 2011. URL: `http://dx.doi.org/10.1137/070697720`, `doi:10.1137/070697720`.

13 Martin E. Dyer, Alan M. Frieze, and Ravi Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991. URL: `http://doi.acm.org/10.1145/102782.102783`, `doi:10.1145/102782.102783`.

14 Ulrich Faigle and György Turán. Sorting and recognition problems for ordered sets. *SIAM J. Comput.*, 17(1):100–113, 1988. URL: `http://dx.doi.org/10.1137/0217007`, `doi:10.1137/0217007`.

15 Wayne Goddard, Claire Kenyon, Valerie King, and Leonard J. Schulman. Optimal randomized algorithms for local sorting and set-maxima. *SIAM J. Comput.*, 22(2):272–283, 1993. URL: `http://dx.doi.org/10.1137/0222020`, `doi:10.1137/0222020`.

16 Anupam Gupta and Amit Kumar. Sorting and selection with structured costs. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 416–425, 2001. URL: `http://dx.doi.org/10.1109/SFCS.2001.959916`, `doi:10.1109/SFCS.2001.959916`.

17 Zhiyi Huang, Sampath Kannan, and Sanjeev Khanna. Algorithms for the generalized sorting problem. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 738–747, 2011. URL: `http://dx.doi.org/10.1109/FOCS.2011.54`, `doi:10.1109/FOCS.2011.54`.

18 Nabil Kahale and Leonard J. Schulman. Bounds on the chromatic polynomial and on the number of acyclic orientations of a graph. *Combinatorica*, 16(3):383–397, 1996. URL: `http://dx.doi.org/10.1007/BF01261322`, `doi:10.1007/BF01261322`.

19 Jeff Kahn and Michael Saks. Balancing poset extensions. *Order*, 1(2):113–126, 1984.

20 János Komlós, Yuan Ma, and Endre Szemerédi. Matching nuts and bolts in o(n log n) time. *SIAM J. Discrete Math.*, 11(3):347–372, 1998. URL: `http://dx.doi.org/10.1137/S0895480196304982`, `doi:10.1137/S0895480196304982`.

21 John E. Savage. *Models of computation - exploring the power of computing*. Addison-Wesley, 1998.