Lower Bounds for Sorted Geometric Queries in the I/O Model

Peyman Afshani[1,⋆] and Norbert Zeh[2,⋆⋆]

[1] MADALGO[⋆⋆⋆], Department of Computer Science
Aarhus University, Denmark, peyman@cs.au.dk
[2] Faculty of Computer Science, Dalhousie University
Halifax, Canada, nzeh@cs.dal.ca

Abstract. We study sorted geometric query problems, a class of problems that, to the best of our knowledge and despite their applications, have not received much attention so far. Two of the most prominent problems in this class are angular sorting queries and sorted K-nearest neighbour queries. The former asks us to preprocess an input point set S in the plane so that, given a query point q, the clockwise ordering of the points in S around q can be computed efficiently. In the latter problem, the output is the list of K points in S closest to q, sorted by increasing distance from q. The goal in both problems is to construct a small data structure that can answer queries efficiently. We study sorted geometric query problems in the I/O model and prove that, when limited to linear space, the naïve approach of sorting the elements in S in the desired output order from scratch is the best possible. This is highly relevant in an I/O context because storing a massive data set in a superlinear-space data structure is often infeasible. We also prove that answering queries using $O(N/B)$ I/Os requires $\Omega(N \log_M N)$ space, where N is the input size, B is the block size, and M is the size of the main memory. This bound is unlikely to be optimal and in fact we can show that, for a particular class of "persistence-based" data structures, the space lower bound can be improved to $\Omega(N^2/M^{O(1)})$. Both these lower bounds are a first step towards understanding the complexity of sorted geometric query problems. All our lower bounds assume indivisibility of records and hold as long as $B = \Omega(\log_{M/B} N)$.

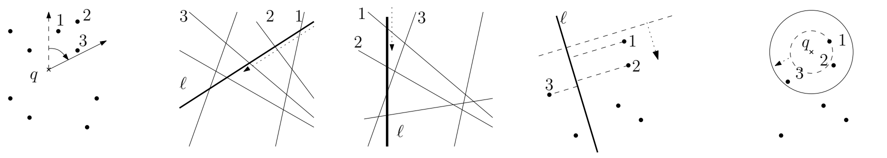
1 Introduction

We study the problem of storing a set of geometric objects in a data structure that supports the following type of queries efficiently: the query implicitly specifies a particular order of the input objects and the data structure has to report

⋆ This research was done while the first author was a postdoctoral fellow at the Faculty of Computer Science of Dalhousie University and was supported by an NSERC postdoctoral fellowship.

⋆⋆ Research supported by NSERC and the Canada Research Chairs programme.

⋆⋆⋆ Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.



(a) Angular sorting (b) Line intersection sorting (c) Ordered line stabbing (d) Line sweep ordering (e) Sorted K-nearest neighbours

Fig. 1. Examples of sorted geometric query problems

the objects in this order. In most of the problems that we study, in contrast to typical data structure problems, the challenge is *not* to decide *which* objects to report: *all* objects in the data structure are to be reported. The challenge is to arrange the objects in the desired order more efficiently than sorting them from scratch. Specifically, we study the following problems.

Angular sorting: Given a set S of points in the plane and a query point q, report the points in S in the order they are passed by a ray shot upwards from q and then rotated in clockwise direction around q (Figure 1(a)).

Line intersection sorting: Given a set S of lines in the plane and a query line ℓ, sort the lines in S by the y-coordinates of their intersections with ℓ (Figure 1(b)). (We assume w.l.o.g. that ℓ is not horizontal and no line in S is parallel to ℓ.)

Ordered line stabbing: This is a special case of line intersection sorting where the query line ℓ is vertical (Figure 1(c)).

Line sweep ordering: Given a set S of points in the plane and a non-horizontal query line ℓ, sort the points in S by the y-coordinates of their projections onto ℓ. This corresponds to the order in which a sweep line perpendicular to ℓ passes over the points in S (Figure 1(d)).

Ordered *K*-nearest neighbours: Given a set S of points in the plane and a query point q, report the K points in S closest to q, sorted by their distances from q (Figure 1(e)).

These arise naturally in a number of geometric algorithms and applications. Angular sorting is used as a preprocessing step in a number of geometric algorithms, e.g., robust statistical measures of a point q in a point set, such as the Tukey depth [13], Oja depth [13] or simplicial depth [7,10,13], can be computed in linear time once the input points are sorted by their angles around q. Angular sorting is also used in the classical convex hull algorithm known as Graham's scan [8] and in the computation of visibility polygons (e.g., see [6,14]). Reporting the K nearest neighbours of a query point is a classical problem with many applications. In a number of these applications, it is natural to ask that these neighbours be reported in order of increasing distance from the query point. For example, when a user of a GPS device asks for the bookstores closest to their current location, it is natural to report the results by increasing distance.

Nevertheless, the existing literature on K-nearest neighbour queries considers exclusively the unordered version of this problem (e.g., see [2]). Line intersection sorting is the dual of angular sorting. Line sweep ordering is motivated by applications of the plane sweep technique in computational geometry.

The basic idea behind these problems is a natural one: many problems can be solved in linear time after an initial sorting step. Thus, presorting the input can help to speed up such algorithms. In one dimension, there is only one ordering of the input, and storing the input in sorted order uses linear space. In two dimensions, the ordering depends on the query and there are sufficiently many different orderings to make the naïve approach of storing each of them explicitly infeasible. On the other hand, the geometry of the input implies that not every ordering of the input objects corresponds to an ordering generated by a query. For instance, N input lines can be intersected by a query vertical line in $O(N^2)$ ways only. Furthermore, the total number of combinatorially different arrangements of lines is $N^{O(N)} = N!^{O(1)}$ (see e.g., [11, Chapter 6]), which is not much larger than the number of permutations of N objects in one dimensions. These raise the question whether the geometric constraints can be exploited to represent all possible output orderings compactly in a data structure while still being able to extract the ordering produced by any query using sub-sorting cost. This is the question we investigate in this paper, from a lower bound perspective.

We focus primarily on ordered line stabbing queries. Using fairly standard reductions, a lower bound for ordered line stabbing queries implies lower bounds for all the problems introduced previously (more details will follow in the full version of the paper).

We study these problems in the I/O model [3]. In this model, the computer has an internal memory capable of holding up to M data items (e.g., points) and an external memory of conceptually unlimited size. Computation has to happen on data in memory and is free. Data is transferred between the two memory levels using *I/O operations* (I/Os). Each such operation transfers a block of B consecutive data elements between internal and external memory. The cost of an algorithm is the number of such I/O operations it performs.

The I/O model is the most widely accepted model for designing algorithms for large data sets beyond the size of the computer's main memory. Statistical data sets can be very large, which motivates the study of methods to speed up the computation of statistical measures in the I/O model. Moreover, any result in the I/O model can also be applied to any two levels in a computer's cache hierarchy, making our results relevant to cache-efficient data structures and thus to data sets of more modest size. The most compelling reason to focus on the I/O model in this paper, however, is that it is the only well-established computing model with known lower bound properties that are sufficiently strong to obtain non-trivial space lower bounds for sorted query problems. Specifically, permuting and sorting have a superlinear cost in the I/O model.

Previous results. The problem of preprocessing an input set S of N points to answer angular sorting queries was considered as early as 1986, in the context of computing the visibility polygon [5]. Their data structure uses $O(N^2)$ space and answers queries in linear time. The techniques used in this structure are now standard: store the arrangement $\mathcal{A}$ formed by the dual of S and answer the query by finding the intersection of $\mathcal{A}$ with the dual line of the query point. Ghodsi and Nouri also studied angular sorting queries [12] but focused on obtaining a *sublinear* query time by returning a pointer to a simple data structure that stores the sorted order. Using N^2h space, for $1 \leq h \leq N^2$, they find such a pointer for any given query in $O((N \log h)/\sqrt{h})$ time. We are not aware of any other non-trivial results for angular sorting. For any fixed K, K-nearest neighbour queries can be answered using $O(\log_B N + K/B)$ I/Os by a linear-space data structure [2][3]; the output can then be sorted using $O((K/B)\log_{M/B}(K/B))$ I/Os.

To the best of our knowledge, the only relevant lower bound result is due to Afshani *et al.* [1], who considered the following one-dimensional problem: given an array A of numbers, preprocess A such that, given two query indices $i < j$, the numbers in the subarray $A[i\,..\,j]$ can be efficiently reported in sorted order. They proved that answering such queries using $O(\log^\alpha n + fK/B)$ I/Os requires $\Omega\left(N\frac{f^{-1}\log_M n}{\log(f^{-1}\log_M n)}\right)$ space, assuming $B = \Omega(\log N)$, where $n = N/B$, f is a parameter with $1 \leq f \leq \log_{M/B} n$, and α is any constant.

Our Results. We prove non-trivial lower bounds for sorted geometric query problems, which as a class of problems are novel. Our results provide the first insights into the complexity of answering such queries.

General lower bounds. Assuming $B = \Omega(\log_{M/B} N)$, we show that, to achieve a query bound of $O(N/B)$ I/Os, any data structure for angular sorting, line sweep ordering, ordered line stabbing or line intersection ordering has to use $\Omega(N \log_M N)$ space in the worst case. To achieve a query bound of $O(\log_B N + K/B)$ I/Os for ordered K-nearest neighbour queries in the plane, for a fixed $K = \Omega(\max(M, B\log_B N))$, $\Omega(N \log_M K)$ space is needed in the worst case. At the other end of the spectrum, we show that linear-space data structures for these problems can only achieve query bounds of $O((N/B)\log_M N)$ and $O(1 + (K/B)\log_M K)$ I/Os, respectively. The latter result is highly relevant in an I/O context because storing a massive data set in a superlinear-space data structure is often infeasible.

Lower bounds for "persistence-based" data structures. Using persistence to build data structures for two- or three-dimensional problems is one of the classical tools in algorithms [9]. For the ordered line stabbing problem, this easily gives a data structure that uses $O(N^2)$ space and can answer queries in optimal time: given an input S of N lines, sweep a vertical line q from $x = -\infty$ to $x = \infty$ and store the order in which the lines in S intersect q in a partially persistent B-tree. Each time q passes the intersection point of two lines in S, only the order of these two lines switches, which results in $O(N^2)$ updates overall and thus in an $O(N^2)$-space data structure [4]. Inspired by this, we consider "persistence-based" data structures that generalize the above solution (for details see Section 3.3). We prove that any persistence-based data structure with

[3] The space requirement increases to $O(N \log^*(N/B))$ if we want to answer K-nearest neighbour queries for any K, where $\log^*$ is the iterated log function.

a linear query bound for angular sorting, line sweep ordering, ordered line stabbing or line intersection ordering requires $\Omega(N^2/M^{O(1)})$ space in the worst case. For 2-d K-nearest neighbour queries, for a fixed $K = \Omega(\max(M, B \log_B N))$, the space bound is $\Omega(NK/M^{O(1)})$. This shows that, if small data structures for these problems exist, more sophisticated techniques are needed to obtain them.

Our lower bound proof is inspired by [1]: the idea is to construct sufficiently many queries that produce sufficiently different permutations of the input that the answers to these queries must be produced using nearly disjoint portions of the data structure if the query bound is to be linear. While the proof in [1] used purely combinatorial properties of integer sequences, we develop a fundamentally different, *geometric* construction of such a set of queries in this paper.

2 Lower Bound Model

The input to any sorted query problem is a set S of objects. Each query implicitly specifies a particular order of S. Our goal is to build a data structure $\mathcal{D}$ that is capable of generating the required ordering of the objects in S, for any query. We view $\mathcal{D}$ as a linear sequence of *cells* (disk locations). Each cell can hold one input object. We assume *indivisibility of records*: The only operations the query algorithm can perform is reading the contents of a block of B cells into memory, rearranging the objects in memory at no cost, and writing B objects currently in memory to a block of cells on disk. The query algorithm *cannot* create new objects. This implies that each object output by the query algorithm can be traced back to an *origin cell* in the data structure. For a query q, we use S_q to denote the output sequence of q. For any subsequence S' of S_q, we use $C(S')$ to denote the set of origin cells used to produce S'.

Our strategy is to show that there exist an input S, a query set $\{q_1, q_2, \dots, q_t\}$, and subsequences $S^m_{q_1}, S^m_{q_2}, \dots, S^m_{q_t}$ of the outputs of these queries such that each sequence $S^m_{q_i}$ has size $\Omega(N)$ and $C(S^m_{q_i}) \cap C(S_{q_j})$ is small, for all $i \neq j$ and any data structure $\mathcal{D}$. Given the right choice of parameters, this implies our lower bounds. Since we only consider the cost of permuting the objects, our lower bound holds even for an omnipotent data structure that is capable of identifying the cheapest set of origin cells for each query at no cost. The following lemma states an important property of this model. A very similar version was proved in [1] and we postpone the proof to the full version of the paper.

Lemma 1. *For a sequence σ of N elements and $1 \leq f \leq \log_M N$, the maximum number of permutations of σ that can be produced using $O(fN/B)$ I/Os is $M^{O(fN)}$, provided $B = \Omega(\log_M N)$.*

3 Lower Bounds for Ordered Query Problems

In this section, we prove our main result (Theorem 1). We prove this theorem for ordered line stabbing and as discussed, standard reductions give the same lower bound for other problems introduced in this paper.

Theorem 1. *For $B = \Omega(\log_{M/B} N)$, any data structure that answers angular sorting, line intersection sorting, line sweep ordering or ordered line stabbing queries using $O(N/B)$ I/Os requires $\Omega(N \log_M N)$ space in the worst case. Any data structure capable of answering ordered K-nearest neighbour queries using $O(\log_B N + K/B)$ I/Os, for any fixed $K = \Omega(\max(M, B \log_B N))$, requires $\Omega(N \log_M K)$ space in the worst case. The query bound of any linear-space data structure for these problems is $\Omega((N/B) \log_M N)$ I/Os or, in the case of K-nearest neighbour queries, $\Omega(\log_B N + (K/B) \log_M K)$ I/Os in the worst case.*

3.1 Input Instance and Geometric Properties

To prove Theorem 1, we construct a random input instance and a query set which, with non-zero probability, require a large part of the output of each query to be produced using a unique set of cells in the data structure (see Section 2). We present this construction in this section and prove a number of geometric properties that are used in Section 3.2 to prove Theorem 1.

Consider two squares $Q := [-N, N] \times [-N, N]$ and $Q^m := [-N/2, N/2] \times [-N/2, N/2]$. The set S of N lines to be stored in the data structure is generated using the following random process. For each line $\ell \in S$, we choose one point each uniformly at random from the left and right sides of Q; ℓ is the line through these two *anchor points*. We create t queries $q_1, q_2, \ldots, q_t$, for a parameter t to be chosen later, by partitioning the top side of Q^m into t subsegments of equal length; $q_1, q_2, \ldots, q_t$ are the vertical lines through the centers of these segments (Figure 2(a)). For each query q_i, let $q_i^m := q_i \cap Q^m$, and let $S_{q_i}^m \subseteq S_{q_i}$ be the sequence of lines in S that intersect q_i^m, sorted by the y-coordinates of their intersections with q_i^m. Our first lemma shows that $S_{q_i}^m$ is large, for each query q_i. From here on, we do not distinguish between a line segment and its length, and we use $I_{\ell,s}$ to denote the event that a random line $\ell \in S$ intersects a segment s.

Lemma 2. *Consider a query q_i and a subsegment $q'_i \subseteq q_i^m$. A random line in S intersects q'_i with probability $\Theta(q'_i/N)$.*

Proof. First we prove that $\Pr[I_{\ell,q'_i}] = \Omega(q'_i/N)$. Assume first that $q'_i \leq N/12$. Then there exists a subsegment s_L of the left side of Q whose length is at least $2N/3 - 4q'_i \geq N/3 = \Omega(N)$ and such that any line intersecting both s_L and q'_i intersects the right side of Q (see Figure 2(b)). Now let $s'_L \subseteq s_L$ be a subsegment of s_L, and let s'_R be the largest subsegment of the right side of Q such that every line with anchors on s'_L and s'_R intersects q'_i (Figure 2(b)). It is easy to verify that, for $s'_L \leq q'_i$, $q'_i \leq s'_R \leq 4q'_i$, that is, $s'_R = \Theta(q'_i)$.

Now consider a random line ℓ in S. Its left anchor lies on s'_L with probability $s'_L/2N$. Its right anchor lies on s'_R with probability $s'_R/2N$. Since a line that intersects both s'_L and s'_R also intersects q'_i, this shows that $\Pr[I_{\ell,s'_L} \cap I_{\ell,q'_i}] \geq s'_L s'_R/(4N^2) = \Omega(s'_L q'_i/N^2)$. Since we can cover s_L with $\lceil s_L/s'_L \rceil$ *disjoint* subsegments of length s'_L, we have $\Pr[I_{\ell,q'_i}] \geq \Pr[I_{\ell,s_L} \cap I_{\ell,q'_i}] = \Omega(s_L q'_i/N^2) = \Omega(q'_i/N)$.

If $N/12 \leq q'_i \leq N$, we apply the same argument to a segment $q''_i \subset q'_i$ of length $N/12 = \Omega(q'_i)$. This proves that $\Pr[I_{\ell,q'_i}] \geq \Pr[I_{\ell,q''_i}] = \Omega(q''_i/N) = \Omega(q'_i/N)$.

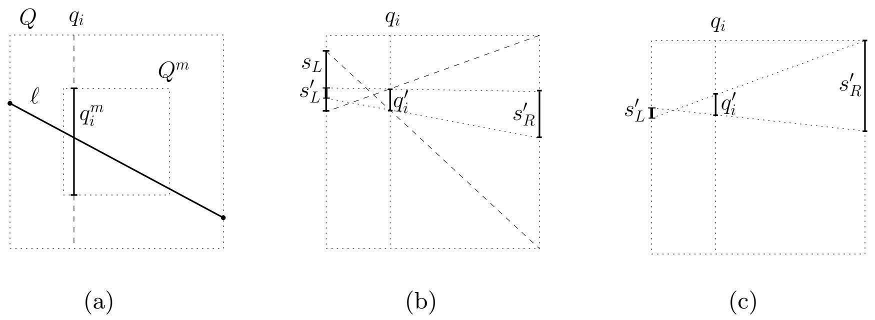


Fig. 2. (a) Q, Q^m, an input line ℓ and a query line q_i. (b,c) Proof of Lemma 2.

The upper bound proof is similar. For any subsegment $s'_L \le q'_i$ of the left side of Q, any line $\ell \in S$ intersecting s'_L and q'_i has its right anchor on a subsegment s'_R of the right side of Q of length at most $4q'_i + 3s'_L = O(q'_i)$ (Figure 2(c)). Thus, $\Pr[I_{\ell,s'_L} \cap I_{\ell,q'_i}] = O(s'_L q'_i/N^2)$. Since we can cover the left side of Q using $\lceil 2N/s'_L \rceil$ subsegments of length s'_L, this proves that $\Pr[I_{\ell,q'_i}] = O(q'_i/N)$. □

Corollary 1. *$|S^m_{q_i}| = \Omega(N)$ with probability at least $1-1/N^c$, for any constant c.*

Proof. By Lemma 2, each line in S intersects q_i^m with constant probability, since q_i^m has length N. Thus, $\mathrm{E}[|S^m_{q_i}|] = \Omega(N)$. The high-probability bound follows by applying Chernoff bounds. □

To prove that any data structure needs to use largely disjoint subsets of cells to produce the sequences $S^m_{q_i}$ and S_{q_j}, we require that S_{q_i} and S_{q_j} are nearly independent random permutations of S. In other words, once we fix the output of query q_i, a large number of permutations remain that are possible outputs of query q_j, provided the distance between q_i and q_j is sufficiently large.

Intuitively, the argument is as follows (see Figure 3(a)): Conditioned on the event that $\ell \in S$ passes through a point $p \in q_i^m$, the intersection between ℓ and q_j is uniformly distributed inside an interval q'_j on q_j (because ℓ intersects the right side of Q). For such a conditional probability to make sense, however, the event it is conditional on must have a non-zero probability, whereas ℓ passes through p with probability zero. To overcome this problem, we consider a subsegment $q'_i \subseteq q_i^m$ instead of a point $p \in q_i^m$ and assume ℓ intersects q'_i (see Figure 3(b)). If q'_i is sufficiently small, this increases the length of q'_j only slightly and the distribution of the intersection between ℓ and q_j over q'_j remains almost uniform:

Lemma 3. *Let $f : q_j \to \mathbb{R}$ be the density function that describes the distribution of the intersection point between a random line $\ell \in S$ and q_j over q_j conditioned on the event that ℓ intersects a subsegment $q'_i \subseteq q_i^m$. Let d be the distance between q_i and q_j. If $q'_i = O(d)$, then f is zero except on a subsegment q'_j of q_j of length $\Theta(d)$, and $f(p) = O(1/d)$, for all $p \in q_j$.*

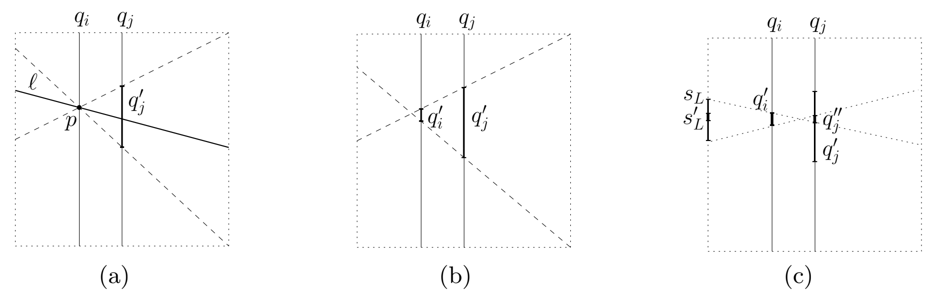


Fig. 3. Independence of $S^m_{q_i}$ and S_{q_j}

Proof. Since every line in S has slope between -1 and 1, the subsegment q'_j of possible intersection points between ℓ and q_j has size at most $q'_i + 2d = O(d)$. Clearly, f is zero outside of q'_j. To bound $f(p)$, for all $p \in q_j$, we consider a subsegment q''_j of q_j such that $q''_j \le q'_i$ (see Figure 3(c)). It suffices to prove that, for any $q''_j > 0$, $\Pr[I_{\ell,q''_j} | I_{\ell,q'_i}] = O(q''_j/d)$. By Lemma 2, $\Pr[I_{\ell,q'_i}] = \Theta(q'_i/N)$. Thus, to prove that $\Pr[I_{\ell,q''_j} | I_{\ell,q'_i}] = O(q''_j/d)$, it suffices to prove that $\Pr[I_{\ell,q'_i} \cap I_{\ell,q''_j}] = O(q'_i q''_j/(dN))$. The ratio between q_i's distance from Q's left side and its distance from q_j is $\Theta(N/d)$. Thus, any line in S intersecting q'_i and q''_j has its left anchor on a subsegment s_L of Q's left side of length at most $(q'_i + q''_j)\Theta(N/d) = \Theta(q'_i N/d)$. Now consider a subsegment $s'_L \subseteq s_L$ with $s'_L \le q''_j$. Line ℓ intersects s'_L, q'_i, and q''_j only if it intersects s'_L and q''_j. The probability of the latter event is $O(s'_L q''_j/N^2)$ because the right anchor of ℓ must be chosen from a subsegment of Q's right side of length at most $3s'_L + 4q''_j = O(q''_j)$ in this case. Now we cover s_L with $\lceil s_L/s'_L \rceil = O(q'_i N/(ds'_L))$ subsegments of length s'_L. The probability that a line $\ell \in S$ intersects both q'_i and q''_j is upper bounded by the probability that it intersects s_L and q'_i, which is $O(q'_i N/(ds'_L) \cdot s'_L q''_j/N^2) = O(q'_i q''_j/(dN))$. □

As we show next, Lemma 3 implies that fixing an order in which the lines in S intersect q_i does not significantly constrain the order in which they intersect q_j.

Lemma 4. *Let $I = [0, d]$ be an interval, and let $f_1, f_2, \ldots, f_X$ be density functions on I such that, for all $1 \le i \le X$ and every $p \in I$, $f_i(p) = O(1/d)$. Let $p_1, p_2, \ldots, p_X$ be random points, where each p_i is chosen according to the density function f_i. For a permutation π of $[X]$, the event E_π that $p_1, p_2, \ldots, p_X$ are sorted according to π happens with probability at most $2^{O(X)}/X!$.*

Proof. By relabeling the points, we can assume that $\pi = \langle 1, 2, \ldots, X\rangle$. E_π is the event that $p_X \in [0, d]$, $p_{X-1} \in [0, p_X]$, $p_{X-2} \in [0, p_{X-1}]$, and so on. For $1 \le j \le X$ and a value $0 \le x \le d$, let $F_j(x)$ be the probability that points $p_1, p_2, \ldots, p_{j-1}$ lie in the interval $[0, x]$ and are sorted according to π. Then

$$\Pr[E_\pi] = \int_0^d f_X(y) F_X(y)\,\mathrm{d}y \quad \text{and} \quad F_j(x) = \int_0^x f_{j-1}(y) F_{j-1}(y)\,\mathrm{d}y.$$

Using induction on j and the facts that $F_1(y) = 1$ and $f_j(y) \leq \alpha/d$, for all $y \in [0, d]$ and some constant $\alpha > 0$, we obtain the bound

$$F_j(y) \leq \frac{(\alpha y/d)^{j-1}}{(j-1)!}, \text{for all } 1 \leq j \leq X.$$

This implies that

$$\Pr[E_\pi] = \int_0^d f_X(y) F_X(y)\,\mathrm{d}y \leq \int_0^d \frac{\alpha}{d} \cdot \frac{(\alpha y/d)^{X-1}}{(X-1)!}\,\mathrm{d}y = \frac{\alpha^X}{X!} = \frac{2^{O(X)}}{X!}. \quad \square$$

Finally, we use Lemmas 3 and 4 to prove a nearly uniform distribution over the possible orders in which (a subset of) the lines in S intersect a query line q_j, even once their intersections with a query line q_i are fixed.

Lemma 5. *Consider two queries q_i and q_j $(i < j)$, and let d be the horizontal distance between q_i and q_j. Let $q'_{i,1}, q'_{i,2}, \ldots, q'_{i,X}$ be pairwise disjoint subsegments of q_i^m of length at most d each, let I be the event $I := I_{\ell_1,q'_{i,1}} \cap I_{\ell_2,q'_{i,2}} \cap \cdots \cap I_{\ell_X,q'_{i,X}}$, where $\{\ell_1, \ell_2, \ldots, \ell_X\} \subseteq S$, and let π be a fixed permutation of the lines $\ell_1, \ell_2, \ldots, \ell_X$. The probability that the lines $\ell_1, \ell_2, \ldots, \ell_X$ intersect q_j in the order specified by π conditioned on event I is at most $\frac{2^{O(X)}}{(Xd/N)^{\Theta(X)}}$.*

Proof. Since $q'_{i,k} \leq d$, for all $1 \leq k \leq X$, and q_i^m has length N, we can partition q_i^m into $\Theta(N/d)$ disjoint subsegments $q''_{i,1}, q''_{i,2}, \ldots, q''_{i,Y}$ of length $\Theta(d)$ and such that either $q'_{i,k} \subseteq q''_{i,h}$ or $q'_{i,k} \cap q''_{i,h} = \emptyset$, for all $1 \leq k \leq X$ and $1 \leq h \leq Y$. Now consider two segments $q'_{i,k} \subseteq q''_{i,h}$ and the line $\ell_k \in S$. By Lemma 3, conditioned on ℓ_k intersecting $q'_{i,k}$, the potential intersection points between ℓ_k and q_j form a subsegment $q'_{j,k}$ of q_j of length $\Theta(d)$, and the distribution of ℓ_k's intersection with q_j over $q'_{j,k}$ is almost uniform. Since $q''_{i,h} = \Theta(d)$, the segments $q'_{j,k}$, for all $q'_{i,k} \subseteq q''_{i,h}$, are contained in a subsegment $q''_{j,h}$ of q_j of length $\Theta(d)$. Now let t_h be the number of segments $q'_{i,k}$ contained in $q''_{i,h}$. By Lemma 4, the lines in S intersecting these segments intersect q_j in the order specified by π with probability at most $2^{O(t_h)}/t_h!$. The lemma follows by observing that the probability that *all* the lines $\ell_1, \ell_2, \ldots, \ell_X$ intersect q_j in the order specified by π is at most $\prod_{h=1}^{Y} \frac{2^{O(t_h)}}{t_h!}$ which is maximized (over the reals) when $t_1 = \cdots = t_Y = X/Y = \Theta(Xd/N)$. $\square$

3.2 Proof of Theorem 1

For any two queries q_i and q_j, let R_{ij} be the event that there exists a data structure such that $C(S_{q_i}^m) \cap C(S_{q_j}) \geq X$, for a parameter X to be chosen later. To bound $\Pr[R_{ij}]$, we partition the segment $q_i \cap Q$ into a set $\mathcal{S}$ of N^4 subsegments of length $2N^{-3}$. Let $\mathcal{S}^N$ be the set of all sequences of N (not necessarily distinct) segments in $\mathcal{S}$. For any $\sigma = \langle s_1, s_2, \ldots, s_N \rangle \in \mathcal{S}^N$, let I_σ be the event that, for all $1 \leq k \leq N$, the kth line $\ell_k \in S$ intersects s_k. For every choice of lines in S, exactly one of the events I_σ, $\sigma \in \mathcal{S}^N$, occurs. Thus, we have

$$\Pr[R_{ij}] = \sum_{\sigma \in \mathcal{S}^N} \Pr[I_\sigma \cap R_{ij}] = \sum_{\sigma \in \mathcal{S}^N} \Pr[I_\sigma] \Pr[R_{ij} | I_\sigma].$$

We bound this sum by partitioning $\mathcal{S}^N$ into two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$. A sequence $\sigma = \langle s_1, s_2, \ldots, s_N \rangle \in \mathcal{S}^N$ belongs to $\mathcal{S}_1$ if $s_h \neq s_k$, for all $1 \leq h < k \leq N$, and, for a small enough constant $\beta > 0$, at least βN of its segments belong to Q^m. Otherwise $\sigma \in \mathcal{S}_2$. By Corollary 1, the sum of the probabilities $\Pr[I_\sigma]$, where σ has less than βN segments in Q^m, is $O(N^{-2})$. The probability that two lines in S intersect the same segment in $\mathcal{S}$ is at most $\binom{N}{2} N^{-4} = O(N^{-2})$. Thus, $\sum_{\sigma \in \mathcal{S}_2} \Pr[I_\sigma] \Pr[R_{ij} | I_\sigma] \leq \sum_{\sigma \in \mathcal{S}_2} \Pr[I_\sigma] = O(N^{-2})$ and

$$\Pr[R_{ij}] = O(N^{-2}) + \sum_{\sigma \in \mathcal{S}_1} \Pr[I_\sigma] \Pr[R_{ij} | I_\sigma].$$

Next we bound $\Pr[R_{ij} | I_\sigma]$, for $\sigma \in \mathcal{S}_1$. For a subset $S' \subseteq S$ of at least X lines, let $R_{S'}$ be the event that the cells in $C(S^m_{q_i}) \cap C(S_{q_j})$ store exactly the lines in S'. Then $\Pr[R_{ij} | I_\sigma] = \sum_{S' \in 2^S, |S'| \geq X} \Pr[R_{S'} | I_\sigma]$. Let S'_i and S'_j be the order of S' as reported by q_i and q_j, respectively, and let $C(S'_i)$ and $C(S'_j)$ be the set of origin cells used to generate S'_i and S'_j, respectively. Event $R_{S'}$ occurs exactly when $C(S'_i) = C(S'_j)$. Thus, if the query cost is $O(fN/B)$ I/Os, it is possible to generate S'_j from S'_i using $O(fN/B)$ I/Os, by going via $C(S'_i)$. Since $\sigma \in \mathcal{S}_1$, the segments in σ are pairwise distinct, that is, no two lines pass through a same segment in σ. Thus, conditioned on event I_σ, S'_i is fixed. Let $\mathcal{P}$ be the set of all possible permutations of S' that can be generated using $O(fN/B)$ I/Os starting from S'_i. Since S'_i is fixed, Lemma 1 shows that $|\mathcal{P}| = M^{O(fN)}$. Given event I_σ, $R_{S'}$ can happen only when $S'_j \in \mathcal{P}$. By Lemma 5, the probability that the lines in S' cross q_j according to any permutation in $\mathcal{P}$ is at most $\frac{2^{O(X)}}{(Xd/N)^{\Theta(X)}}$, where d is the distance between q_i and q_j. Thus, $\Pr[R_{S'} | I_\sigma] \leq \frac{2^{O(X)} M^{O(fN)}}{(Xd/N)^{\Theta(X)}}$. Since there are at most 2^N different subsets S', $\Pr[R_{ij} | I_\sigma] \leq 2^N \cdot \frac{2^{O(X)} M^{O(fN)}}{(Xd/N)^{\Theta(X)}} = \frac{2^{O(X)} M^{O(fN)}}{(Xd/N)^{\Theta(X)}}$. If we choose $X = \frac{\gamma f N}{\log_M N}$, for a large enough constant γ and $d \geq N^\varepsilon$, for a fixed constant $\varepsilon > 0$, it follows that

$$\Pr[R_{ij} | I_\sigma] = \frac{2^{O(X)} M^{O(fN)}}{(Xd/N)^{\Theta(X)}} \leq \frac{2^{O(fN \log M)}}{2^{\gamma \cdot \Omega(fN \log N / \log_M N)}} \leq \frac{2^{O(fN \log M)}}{2^{\gamma \Omega(fN \log M)}} = O(N^{-2})$$

and

$$\Pr[R_{ij}] = O(N^{-2}) + \sum_{\sigma \in \mathcal{S}_1} O\left(\frac{\Pr[I_\sigma]}{N^2}\right) = O(N^{-2}).$$

Thus, for $f = 1$ and $t = (\log_M N)/(2\gamma) = (N/2)/X$ queries at distance at least N^ε from each other, we have a non-zero probability that each query uses at least $N/2$ origin cells not used by any other query. This proves that any ordered line stabbing data structure with query bound $O(N/B)$ has to use $\Omega(tN) = \Omega(N \log_M N)$ space. For $f = o(\log_M N)$ and $t = \log_M N/(2\gamma f) = \omega(1)$ queries, we once again have a non-zero probability that each query uses at least $N/2$ origin cells not used by any other query. This proves that any data structure with query bound $o((N/B) \log_M N)$ must use $\Omega(tN) = \omega(N)$ space.

3.3 Lower Bound for Persistence-Based Data Structures

As discussed in the introduction, by using a partially persistent B-tree in a straightforward manner, it is possible to answer ordered line stabbing queries using $O(N/B)$ I/Os and $O(N^2)$ space. Here we prove that one cannot do much better, even using a more complex application of (full) persistence.

We consider the following model for persistence. Let $q_1, q_2, \ldots, q_t$ be a sequence of all the possible queries in an arbitrary order. A "persistence-based" structure is built in t stages. At the beginning of stage i the data structure stores a number of *versions*, $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_{i-1}$, each of which is a particular permutation of the input elements. The query q_j can be answered by retrieving $\mathcal{V}_j$ and transforming it into S_{q_j} using $O(N/B)$ I/Os. During stage i, we pick some version $\mathcal{V}_j$, $j < i$, and perform some number $k \geq 0$ of updates (insertions and deletions) on it to obtain a new version $\mathcal{V}_i$ that can be transformed into S_{q_i} using $O(N/B)$ I/Os. This new version is represented by storing only the k updates required to transform $\mathcal{V}_j$ into $\mathcal{V}_i$. The final space consumption is thus the total number of updates required to answer all queries, which is a lower bound on the size of any realistic persistent data structure that can answer queries $q_1, q_2, \ldots, q_t$.

Our lower bound exploits the property of the above model that every query q_i can share origin cells with at most one previous query q_j. (It may share origin cells also with other queries, but these are a subset of the cells it shares with q_j.)

Theorem 2. *Any persistence-based data structure with a query bound of $O(\frac{N}{B})$ I/Os for ordered line stabbing requires $\Omega(N^2/M^{O(1)})$ space.*

Proof. We follow the proof of Theorem 1 in Section 3.2 but choose different parameters. We pick $d = 4M^\gamma$, for a constant γ, $X = N/4$, and $f = 1$. We have,

$$\Pr[R_{ij}|I_\sigma] = \frac{2^{O(X)}M^{O(fN)}}{(Xd/N)^{\Theta(X)}} \leq \frac{2^{O(N\log M)}}{(M^\gamma)^{\Theta(X)}} = \frac{2^{O(N\log M)}}{2^{\Omega(\gamma N\log M)}} = O(N^{-2}).$$

Thus, with non-zero probability, every query q_i shares at most $N/4$ origin cells with any previous query q_j; so to obtain a version $\mathcal{V}_i$ that answers q_i using $O(N/B)$ I/Os, the data structure needs to perform at least $3N/4$ updates on $\mathcal{V}_j$, regardless of the choice of $\mathcal{V}_j$. Since $d = 4M^\gamma$, we can construct a set of $N/(4M^\gamma)$ queries with pairwise distance at least d. Each such query q_i contributes at least $\Omega(N)$ to the size of the data structure, the data structure has size $\Omega(N^2/M^\gamma)$. □

4 Conclusions

We provided the first space lower bounds for a number of sorted geometric query problems. We focused on the I/O model to exploit the non-trivial nature of permuting in this model as a basis for our results. Proving similar lower bounds in other models, such as pointer machine or RAM, is likely to be very difficult, as no suitable superlinear sorting lower bounds are known in these models.

We believe it is unlikely that any of the problems we studied admits an $O(N^{2-\varepsilon})$ space solution. Thus, a natural question is whether our lower bounds can be strengthened. There are a number of barriers that make this very difficult to achieve. For example, any two queries q_i and q_j can share $\Theta(N/\log_M N)$ origin cells, as sorting $\Theta(N/\log_M N)$ elements takes $O(N/B)$ I/Os. This makes our general $\Omega(N \log_M N)$ space lower bound difficult to improve without developing fundamentally different and substantially more difficult techniques. Similarly, it is not difficult to see that our $\Omega(N^2/M^{O(1)})$ space lower bound for persistence-based data structures is actually optimal.

Acknowledgements. We would like to thank Timothy Chan for bringing the type of problems we study in this paper to our attention.

References

1. P. Afshani, G. Brodal, and N. Zeh. Ordered and unordered top-K range reporting in large data sets. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms*, pages 390–400, 2011.
2. P. Afshani and T. M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 180–186, 2009.
3. A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.
4. L. Arge, A. Danner, and S.-M. Teh. I/O-efficient point location using persistent B-trees. In *Proceedings of the 5th Workshop on Algorithm Engineering and Experiments*, pages 82–92, 2003.
5. T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1:49–63, 1986.
6. B. Chazelle. Filtering search: a new approach to query answering. *SIAM Journal on Computing*, 15(3):703–724, 1986.
7. J. Gil, W. Steiger, and A. Wigderson. Geometric medians. *Discrete Mathematics*, 108(1-3):37–51, 1992.
8. R. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.
9. H. Kaplan. Persistent data structures. In *Handbook on Data Structures and Applications.* CRC Press, 2005.
10. S. Khuller and J. S. B. Mitchell. On a triangle counting problem. *Information Processing Letters*, 33(6):319–321, 1990.
11. J. Matoušek. *Lectures on Discrete Geometry.* Springer-Verlag, 2002.
12. M. Nouri and M. Ghodsi. Space-query-time tradeoff for computing the visibility polygon. In *Proceedings of the 3rd International Workshop on Frontiers in Algorithmics*, volume 5598 of *Lecture Notes in Computer Science*, pages 120–131. Springer-Verlag, 2009.
13. P. J. Rousseeuw and I. Ruts. Bivariate location depth. *Journal of Applied Statistics*, 45(4):516–526, 1996.
14. S. Suri and J. O'Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. In *Proceedings of the 2nd ACM Symposium on Computational Geometry*, pages 14–23. ACM, 1986.