On Constant Factors in Comparison-Based Geometric Algorithms and Data Structures

Timothy M. Chan* Patrick Lee†

March 20, 2014

Abstract

Many standard problems in computational geometry have been solved asymptotically optimally as far as comparison-based algorithms are concerned, but there has been little work focusing on improving the constant factors hidden in big-Oh bounds on the number of comparisons needed. In this paper, we consider orthogonal-type problems and present a number of results that achieve optimality in the constant factors of the leading terms, including:

- an algorithm for the 2D maxima problem that uses $n \lg h + O(n\sqrt{\lg h})$ comparisons, where h denotes the output size;
- a randomized algorithm for the 3D maxima problem that uses $n \lg h + O(n \lg^{2/3} h)$ expected number of comparisons;
- a randomized algorithm for detecting intersections among a set of orthogonal line segments that uses $n \lg n + O(n\sqrt{\lg n})$ expected number of comparisons;
- a data structure for point location among 3D disjoint axis-parallel boxes that can answer queries in $(3/2) \lg n + O(\lg \lg n)$ comparisons;
- a data structure for point location in a 3D box subdivision that can answer queries in $(4/3) \lg n + O(\sqrt{\lg n})$ comparisons.

Some of the results can be adapted to solve nonorthogonal problems, such as 2D convex hulls and general line segment intersection.

Our algorithms and data structures use a variety of techniques, including Seidel and Adamy's planar point location method, weighted binary search, and height-optimal BSP trees.

1 Introduction

Asymptotic bounds on the number of comparisons required for classical problems such as sorting and selection are well understood. Researchers have even investigated the constant factors in the leading term for the exact worst-case number of comparisons. For example, the optimal number of comparisons required to sort n numbers [33, 24] is $n \lg n - \Theta(n)$, i.e., the constant factor in the leading term is 1. For median finding [20, 21], currently the best upper bound on the worst-case number of comparisons is about $2.95n$ and the best lower bound is about $(2+\delta)n$ for some concrete

*Cheriton School of Computer Science, University of Waterloo, tmchan@uwaterloo.ca. This work was supported by NSERC; part of the work was done during the author's visit to the Department of Computer Science and Engineering, Hong Kong University of Science and Technology.

†Cheriton School of Computer Science, University of Waterloo, p3lee@uwaterloo.ca.

constant $\delta > 0$. For randomized median-finding algorithms [18], both the upper and lower bound on the expected number of comparisons are $1.5n \pm o(n)$. For heap building [27, 35], the best upper bound is $1.625n + O(\lg n \lg^* n)$ and the best lower bound is about $1.37n$. The list goes on.

By contrast, not much work has been done regarding constant factors of comparison-based algorithms in the computational geometry literature. The present paper investigates this research direction. There are obviously a limitless number of problems in our field that one might consider, not all are worth pursuing from this new perspective, but we will pick several representative problems that have fundamental importance and that require nontrivial interesting techniques to find the best constant factors, not just straightforward modifications of existing algorithms. We choose mainly orthogonal-type problems where the obvious definition of a "comparison" suffices—comparing of two coordinate values of two input points, or in the case of a data structure problem, comparing the coordinate value of a query point with an input point.

Results. The highlights of our results are listed below:

1. We can compute the h maxima among a set of n points in 2D by an output-sensitive algorithm that uses $n \lg h + O(n\sqrt{\lg h})$ comparisons (see Section 2.1 or 2.2).

2. We can compute the maxima among a set of n points in 3D by a randomized algorithm that uses $n \lg n + O(n\sqrt{\lg n})$ expected number of comparisons (see Section 2.3). By contrast, the previous algorithm requires about $2n \lg n$ comparisons.

3. We can compute the h maxima among a set of n points in 3D by a randomized output-sensitive algorithm that uses $n \lg h + O(n \lg^{2/3} h)$ expected number of comparisons (see Section 2.4).

4. We can detect whether a set of n line segments, each vertical or horizontal, by a randomized algorithm that uses $n \lg n + O(n\sqrt{\lg n})$ expected number of comparisons (see Section 2.5). By contrast, the trivial algorithm requires $3n \lg n$ comparisons. For reporting all k intersections, the number increases by an $O(k)$ (or better) term. The same result holds for computing vertical decomposition of n horizontal line segments.

5. We can build a data structure for a set of n disjoint boxes in 3D so that we can locate the box containing a query point using $(3/2) \lg n + O(\lg \lg n)$ comparisons (see Section 3.3).

6. We can build a data structure for a subdivision of space into n disjoint boxes in 3D so that we can locate the box containing a query point using $(4/3) \lg n + O(\lg \lg n)$ comparisons (see Section 3.4).

All of the above bounds are optimal except for the lower-order terms (see Appendix A for lower bound proofs for items 1 and 4). Items 5–6 may be viewed as 3D generalizations of a result by Seidel and Adamy [43], who previously showed that 2D point location queries can be answered using $\lg n + O(\sqrt{\lg n})$ comparisons.

Some of these results may be extended to nonorthogonal problems; for example, item 1 also applies to 2D output-sensitive convex hulls, item 3 applies to 3D convex hulls, and item 4 applies to general 2D line segment intersection and vertical decomposition of line segments. However, for nonorthogonal problems, the definition of a "comparison" needs to be extended to include testing of certain predicates (e.g., determining whether three points are in clockwise order), and we get into the delicate issue of whether just counting the number of predicate tests is meaningful, or whether

some predicates should cost more than others (related is the nontrivial issue of which predicates are actually necessary, also addressed in a series of work, starting with [36], on geometric algorithms that use minimum-degree predicates).

Motivation. Improving constant factors in the running time of asymptotically optimal algorithms of course is important in practice, and the number of comparisons is a measure that can be formally studied from the theoretical perspective. We do not claim practicality of most of our algorithms, however, because of potentially larger constants in not only the lower-order terms but also the cost of other operations beside comparisons (although in all our algorithms the actual running time will be proportional to the stated bounds on the number of comparisons). One exception is our data structure results: here, the query algorithms are simple enough to make the number of comparisons a good indicator of the actual worst-case query time.

One could concoct hypothetical scenarios in which performing comparisons or "probes" to the input is expensive, making it important for algorithms to minimize the number of comparisons. But honestly it just seems a natural question to ask what is the minimum number of comparisons required to solve basic geometric problems. Such pursuit may lead to deeper understanding of the complexity of these problems and the relative power of different algorithmic techniques. Besides, the comparison model has long been recognized as a fruitful model to study lower bounds (although the main contributions of the present paper lie on the upper bound side).

The research direction here may be viewed as complementary to the recent direction in exploring non-comparison-based word-RAM algorithms in computational geometry (e.g., [10]). Ironically, ideas from work on these algorithms will turn out to be useful also for comparison-based algorithms (see Section 2.3). Another recent research direction on revisiting fundamental problems in computational geometry concerns instance-optimal algorithms [1]. These algorithms are comparison-based, and may use potentially fewer comparisons than the bounds guaranteed by our results on "easier" instances; the issue of constant-factor overhead was not addressed there, though. Related are distribution-sensitive data structures, and here some previous papers did take in account constant factors (e.g., see Arya et al.'s work [2] which builds on Seidel and Adamy's [43]).

More on previous related work. The prior work which is closest to ours, and which we have already mentioned, is Seidel and Adamy's paper on planar point location [43]. They actually looked beyond the leading term and showed how to answer queries in $\lg n + 2\sqrt{\lg n} + O(\lg \lg n)$ comparisons, and they proved a nearly matching lower bound in some restricted model. (Comparisons includes x-comparisons of vertices, and vertex/edge aboveness tests.) The space of the data structure is superlinear, but can be made linear at the expense of increasing the third term of the query bound to $O(\lg^{1/4} n)$.

Along the same vein was a lesser known paper by Dubé [22] back in SoCG'93, who studied the problem of d-dimensional dominance emptiness queries: decide whether a query orthant contain a data point, and if so, report any such point. He gave a data structure to answer queries using about $\lfloor d/2 \rfloor \lg n$ comparisons (he didn't explicitly state the lower-order term). By contrast, the trivial algorithm requires $d \lg n$ comparisons. He also proved a matching lower bound. Alternatively, for even d, a technique by Chan [7] for Klee's measure problem can be adapted to give a simple method for dominance emptiness with $(d/2) \lg n + O(1)$ comparisons.

Several previous work on the 2D maxima problem analyzed constant factors but only for the special setting of uniformly distributed points inside a square. Golin [26] and Clarkson [16] de-

scribed algorithms using $n + O(n^{6/7} \log^5 n)$ and $n + O(n^{5/8})$ expected number of point comparisons respectively. One "point comparison" apparently involves two coordinate comparisons.

There are other isolated papers touching on the issue of constant factors. A notable early example is Fibonacci search, which can be used to compute the intersection of a convex polygon with a line [12] using $2 \log_\phi n + O(1)$ slope comparisons where ϕ is the golden ratio.

Techniques. Seidel and Adamy's data structure [43] is a central tool we use to obtain many of our algorithmic results, along with *random sampling* techniques. The ideas behind our 3D data structures are also inspired by Seidel and Adamy's (namely their use of *weighted search tree* techniques with cleverly defined weights).

For our output-sensitive algorithms, the commonly seen *guessing trick* needs refinement, in order to avoid a constant-factor increase. For our 3D point location data structures, we will see that the problem amounts to finding new constructions of *orthogonal binary space partition (BSP)*, not just with good size bounds (as are typically studied) but with good height bounds.

2 Offline Problems

2.1 Output-Sensitive Maxima in 2D

In the *maxima* problem, we are given a set S of n points and we want to compute all maximal points, i.e., all points $p \in S$ such that p is not dominated by any other point in S. (Point $q = (q_1, \ldots, q_d)$ *dominates* point $p = (p_1, \ldots, p_d)$ if $q_j > p_j$ for every j.)

To warm up, we begin with the 2D maxima problem. Here, the maxima form a monotone chain, i.e., a "staircase". A simple, standard algorithm [34, 42] already uses $n \lg n + O(n)$ comparisons: after sorting the x-coordinates, the problem can be solved easily in linear time by scanning the points from right to left. The leading term is worst-case optimal in n. We therefore turn our attention to output-sensitive algorithms.

Kirkpatrick and Seidel at the very first SoCG [31] presented an output-sensitive algorithm for the 2D maxima problem with running time $O(n \lg h)$, where h denotes the number of maximal points. This algorithm is related to their more well-known output-sensitive algorithm for 2D convex hulls [32].

Their maxima algorithm is simple: divide the point set into two subsets by using the median of the x-coordinates, prune all points in the left subset that are below the highest point in the right subset, then recursively compute the maxima of the two subsets. However, the constant factor in the $O(n \lg h)$ bound is strictly greater than 1, because of the cost of median finding as well as pruning.

To improve the constant factor, we propose a variant of Kirkpatrick and Seidel's algorithm with a larger branching factor b, which grows as the level in the recursion increases. The details are given below. Our analysis is based on ideas from Chan, Snoeyink, and Yap [11].

Algorithm outline.

1. divide S into subsets $S_1, \ldots, S_b$ by using $b - 1$ quantiles of the x-coordinates
2. for each subset S_j in right-to-left order:
3. prune all points from S_j that are lower than the highest maximal point computed so far
4. recursively compute the maxima of S_j

Analysis. Line 1 requires solving a *multiple selection* problem for a set of numbers in one dimension (the x-coordinates). A randomized algorithm by Kaligosi et al. [30] solves this problem with a near-optimal expected number of comparisons; in the case of $b-1$ uniformly spaced ranks, the number is $n \lg b + O(n)$.[1] Line 3 requires $O(n)$ comparisons. One can now see the advantage of choosing a nonconstant b; even though we do not know how to optimize the hidden constant factor in the $O(n)$ term, the $n \lg b$ term will dominate.

To be specific, one way is to let $r_i = 2^{i^2}$ and choose $b = r_i/r_{i-1}$ at the i-th level of recursion. (Here, $b = 2^{\Theta(i)}$, so the branching factor grows exponentially in the level i.) Let ℓ satisfy $r_{\ell-1} < h \lg h \le r_\ell$; then $\ell < \sqrt{\lg(h \lg h)} + 1$.

The i-th level of the recursion has at most r_{i-1} subproblems of size at most n/r_{i-1}. The total expected number of comparisons at the i-th level can thus be bounded by $r_{i-1} \cdot [(n/r_{i-1}) \lg(r_i/r_{i-1}) + O(n/r_{i-1})] = n \lg(r_i/r_{i-1}) + O(n)$. By a telescoping sum, the total over the first ℓ levels is at most $n \lg r_\ell + O(n\ell) = n\ell^2 + O(n\ell) = n \lg h + O(n\sqrt{\lg h})$.

On the other hand, the i-th level can have at most h nontrivial subproblems. The total expected number of comparisons at the i-th level can thus alternatively be bounded by $h \cdot [(n/r_{i-1}) \lg(r_i/r_{i-1}) + O(n/r_{i-1})] = O(h(n/r_{i-1}) \lg r_i)$. By a geometric series, the total over all levels greater than ℓ is $O(h(n/r_\ell) \lg r_{\ell+1}) = O(n)$.

We conclude that the overall expected number of comparisons is $n \lg h + O(n\sqrt{\lg h})$.

2.2 Output-Sensitive Maxima in 2D: Bottom-Up Version

In this subsection, we note an alternative output-sensitive algorithm for the 2D maxima problem by adapting Chan's output-sensitive algorithm for 2D convex hulls [5]. The approach here can be viewed as a bottom-up (mergesort-like) instead of top-down (quicksort-like) divide-and-conquer.

Define an *r-grouping* to be any partitioning of S into n/r groups of size r, together with the staircase (i.e., the sorted list of maxima) of each group.

Lemma 2.1. *Given an r-grouping, we can compute the h maxima of S in $O(h(n/r) \lg r)$ time.*

Proof. We use an algorithm similar to Jarvis march [29]. Given a current maximal point q, we describe an $O((n/r) \lg r)$-time algorithm to find the next maximal point q' (to the right of q). Then all maxima can be generated from left to right by performing h such operations.

To perform such an operation, we find the highest point to the right of q in each group, by a binary search in the staircase of the group. The cost of these n/r binary searches is indeed $O((n/r) \lg r)$. Then q' is simply the highest of these points found over all groups. □

An r-grouping can be computed using $(n/r) \cdot (r \lg r + O(r)) = n \lg r + O(n)$ comparisons. If h is known, we could set $r = h \lg h$, compute an r-grouping in $n \lg h + O(n \lg \lg h)$ comparisons, and then compute the h maxima in $O(n)$ additional comparisons by the above lemma. As h is not given in advance, the original version of Chan's algorithm [5] applies a "guessing" trick, using an increasing sequence of r values; unfortunately, this hurts the constant factor. To improve the constant factor, the idea is to compute an r-grouping not from scratch but by merging groups from the previous r-grouping.

[1] Kaligosi et al.'s deterministic algorithm [30] has a larger lower-order term, with $n \lg b + O(n \lg b \lg \lg \lg b / \lg \lg b)$ comparisons. For an alternative solution, we can divide the input into n/s groups of size s and sort each group in total $(n/s) \cdot (s \lg s + O(s))$ number of comparisons; as Frederickson and Johnson [25] described a selection algorithm for n/s sorted arrays of size s with $O((n/s) \lg s)$ running time, b selections take $O(b(n/s) \lg s)$ time. Choosing $s = b \lg b$ gives a deterministic multi-selection algorithm using $n \lg b + O(n \lg \lg b)$ comparisons.

Lemma 2.2. *Given an r-grouping and $r' > r$, we can compute an r'-grouping in $n \lg(r'/r) + O(n)$ comparisons.*

Proof. It suffices to show how to merge the staircases of r'/r groups of size r in $r' \lg(r'/r) + O(r')$ comparisons. Since two sorted lists of total size r' can be merged in at most r' comparisons, r'/r sorted lists of total size r' can be merged by divide-and-conquer in at most $r'\lceil \lg(r'/r) \rceil$ comparisons. Once the union of the r'/r groups has been sorted by x-coordinates, its staircase can be found in a linear number of comparisons by the standard linear-scan algorithm. □

Algorithm outline.

1. for $i = 1, 2, \ldots$ until all maxima are found:
2. compute an r_i-grouping from the previous r_{i-1}-grouping by Lemma 2.2
3. try to compute the maxima of S using the r_i-grouping by Lemma 2.1, if $h \lg h \le r_i$

Analysis. We choose the sequence $r_i = 2^{i^2}$. Let ℓ satisfy $r_{\ell-1} < h \lg h \le r_\ell$; then $\ell < \sqrt{\lg(h \lg h)} + 1$.

Line 2 takes at most $n \lg(r_i/r_{i-1}) + O(n)$ comparisons. By a telescoping sum, the total over all iterations is at most $n \lg r_\ell + O(n\ell) = n\ell^2 + O(n\ell) = n \lg h + O(n\sqrt{\lg h})$.

Line 3 takes $O(n)$ comparisons, because we can declare the trial a failure as as soon as the algorithm has discovered enough maximal points to violate $h \lg h \le r_i$. The total over all iterations is $O(n\ell) = O(n\sqrt{\lg h})$.

We conclude that the overall number of comparisons is $n \lg h + O(n\sqrt{\lg h})$.

Remarks. The choice of sequence $r_i = 2^{i^2}$ and the idea of merging groups from the previous grouping were already remarked upon in [6], though the constant factors were not analyzed there.

Both algorithms can be modified to solve the 2D convex hull problem using a similar number of comparisons, although the definition of a "comparison" needs to be extended to include sidedness tests and other predicates. The number of x-comparisons is $n \lg h + O(n\sqrt{\lg h})$, but the number of other comparisons is actually $O(n)$.

2.3 Maxima in 3D

Next we tackle the more challenging 3D maxima problem. The trivial algorithm uses $3n \lg n + O(n)$ comparisons: we just sort all x-, y-, and z-coordinates; afterwards, no more comparisons are needed. A standard sweep-based algorithm [34, 42] uses $2n \lg n + O(n)$ comparisons: we first sort the z-coordinates in $n \lg n + O(n)$ comparisons, and then insert points in decreasing z-order; each insertion requires at most $\lg n$ comparisons by binary search in the xy-projected staircase.

A still better algorithm can be obtained by a different approach, exploiting a powerful tool by Seidel and Adamy [43]—that planar point location queries require $\lg n + O(\sqrt{\lg n})$ comparisons. The reduction from 3D maxima to 2D point location uses Clarkson–Shor-style sampling [17]. A similar reduction from 3D convex hulls to 2D point location was noted before by Chan and Pătraşcu [10] but in the context of non-comparison-based, word-RAM algorithms. We notice that this reduction can also help in reducing the number of comparisons.

Before describing the algorithm, we first restate the maxima problem in a more convenient form. Instead of a point set, suppose that we are given a set S of orthants, each of the form

$(-\infty, x] \times (-\infty, y] \times (-\infty, z]$. Define the "staircase" polyhedron $\mathcal{P}(S)$ to be union of the orthants of S; the polyhedron has $O(|S|)$ vertices, edges, and faces. The original problem is equivalent to identify the orthants of S whose corners are vertices of $\mathcal{P}(S)$. We will solve a more general problem—namely, construct all the vertices, edges, and faces of $\mathcal{P}(S)$.

Let $\mathrm{VD}(S)$ denote the cells in the *vertical decomposition* of the complement of (i.e., the region above) $\mathcal{P}(S)$. The decomposition is defined as follows: take each horizontal face (a polygon) of $\mathcal{P}(S)$ and subdivide it into rectangles by adding y-vertical line segments at its vertices; then extend each resulting rectangle upward to $z = \infty$.

Algorithm outline.

1. take a random sample $R \subset S$ of size r
2. build a point location structure for $\mathrm{VD}(R)$
3. for each $s \in S$:
4. locate the cell $\Delta \in \mathrm{VD}(R)$ that contains the corner of s
5. find all cells intersecting s by walking in $\mathrm{VD}(R)$, starting from Δ
6. for each $\Delta \in \mathrm{VD}(R)$:
7. obtain the *conflict list* $S_\Delta = \{s \in S : s \text{ intersects } \Delta\}$
8. solve the problem for S_Δ inside Δ

Analysis. We choose $r = n/\lg n$. Line 2 takes $O(r \lg r) = O(n)$ expected number of comparisons, since we can construct $\mathcal{P}(R)$ by the standard sweep algorithm for 3D maxima, generate $\mathrm{VD}(R)$ from $\mathcal{P}(R)$ in linear time, and apply Seidel and Adamy's (randomized) preprocessing algorithm [43].

Line 4 takes a total of $n \cdot (\lg n + O(\sqrt{\lg n}))$ comparisons by n invocations of Seidel and Adamy's query algorithm [43].

Line 5 takes time linear in the sum of the degrees of the cells intersecting s, by breadth- or depth-first search, where the degree δ_Δ of a cell Δ refers to the number of neighboring cells in $\mathrm{VD}(R)$. The total number of comparisons in lines 5 (and 7) is thus $O\left(\sum_{\Delta \in \mathrm{VD}(R)} \delta_\Delta |S_\Delta|\right)$. By Clarkson and Shor's analysis [17],[2] the expected value of this expression is $O(r \cdot n/r) = O(n)$. Line 8 takes $O\left(\sum_{\Delta \in \mathrm{VD}(R)} |S_\Delta| \lg |S_\Delta|\right)$ comparisons if the subproblems are solved directly by the standard sweep algorithm. By Clarkson and Shor's analysis [17], the expected value of this expression is $O(r \cdot (n/r) \lg(n/r)) = O(n \lg \lg n)$.

We conclude that the overall expected number of comparisons is $n \lg n + O(n\sqrt{\lg n})$.

2.4 Output-Sensitive Maxima in 3D

We now present an output-sensitive algorithm for the 3D maxima problem, the main result of the first part of the paper.

Extending the top-down divide-and-conquer approach in Section 2.1 to 3D is not obvious. Kirkpatrick and Seidel [31] described more than one output-sensitive algorithm for the maxima problem in 3 and higher dimensions, but the hidden constant factors are all strictly greater than 1. Clarkson and Shor [17] presented an $O(n \lg h)$ randomized algorithm for 3D convex hulls (later derandomized by Chazelle and Matoušek [14]), which can be adapted for the 3D maxima problem

[2] To deal with the occurrence of δ_Δ, one needs a more carefully defined configuration space when applying their technique; e.g., see Lemma 4.2 of [17].

and can be regarded as a generalization of the division strategy from Section 2.1. At the i-th level, we divide into $O(r_{i-1})$ subproblems of roughly $O(n/r_{i-1})$ size by using conflict lists S_Δ for the cells $\Delta \in \mathrm{VD}(R)$ for a random sample $R \subset S$. We prune away trivial subproblems to ensure that at most h subproblems are generated at each level. The pruning step in Clarkson and Shor's algorithm required executing a 2D $O(n \lg h)$ algorithm along cell boundaries, causing an increase in the constant factor.

Alternatively, the bottom-up approach in Section 2.2 by Chan [5] can be extended to 3D, but the "guessing" of the output size causes a constant-factor blow-up, because we do not know how to merge two staircase polyhedra in linear time with constant factor 1. Yet another $O(n \lg h)$ algorithm for 3D maxima proposed by Clarkson [16] (see also [6, 38]) employs an incremental approach; this too has constant factor strictly greater than 1.

Our solution follows the top-down approach using Clarkson–Shor-style random sampling, but instead of pruning by solving 2D subproblems, we borrow an idea from the bottom-up approach (as in Section 2.2), of running a Jarvis-march-like algorithm at every level of recursion. This combination of ideas is new.

Define an *r-division* to consist of the following: (i) a random sample $R \subset S$ of size r, (ii) a point location structure for $\mathrm{VD}(R)$ with $O(\lg r)$ query time, and (iii) the conflict list S_Δ for every $\Delta \in \mathrm{VD}(R)$.

Lemma 2.3. *Given an r-division, we can compute the h maxima of S in $O(h(n/r)\lg^2 r)$ expected number of comparisons.*

Proof. We use an algorithm similar to Jarvis march or gift wrapping [29, 42], to compute the staircase polyhedron $\mathcal{P}(S)$. Given a current vertex q of $\mathcal{P}(S)$, we describe an $O((n/r)\lg^2 r)$-time algorithm to find the neighboring vertex of q in $\mathcal{P}(S)$ along each of the at most 6 directions; in other words, to determine the first point on the boundary of $\mathcal{P}(S)$ hit by an axis-parallel ray from q. Then the entire polyhedron $\mathcal{P}(S)$ can be generated by a breadth- or depth-first search using $O(h)$ such *ray shooting* operations.

By binary search over the $O(r)$ coordinates of R, a ray shooting operation reduces to $O(\lg r)$ number of *segment emptiness* operations: given an axis-parallel segment qq', decide if the intersection of qq' and the interior of $\mathcal{P}(S)$ is empty. Since $\mathcal{P}(S)$ is orthogonally convex and q is outside the interior of $\mathcal{P}(S)$, segment emptiness is equivalent to *membership testing*: does a point q' lie in the interior of $\mathcal{P}(S)$?

To perform such a membership test, we first locate the cell $\Delta \in \mathrm{VD}(R)$ containing q' in $O(\lg r)$ time (we may assume that q' is in the complement of $\mathcal{P}(R)$, for otherwise the answer is no). We can then test whether q' lie inside any of the orthants in S_Δ. The cost is $O(\max_{\Delta \in \mathrm{VD}(R)} |S_\Delta|)$. By Clarkson's analysis [15], this expression has expected value $O((n/r)\lg r)$. □

It is possible to compute an r-division in $n \lg r + O(n\sqrt{\lg r})$ comparisons by the approach from Section 2.3. If h is known, we could set $r = h \lg^2 h$, compute an r-division in $n \lg h + O(n\sqrt{\lg h})$ comparisons, and then compute the h maxima in $O(n)$ additional comparisons by the above lemma. As h is not given in advance, we need to apply a "guessing" trick, using an increasing sequence of r values; unfortunately, this hurts the constant factor. To improve the constant factor, the idea is to compute an r-division not from scratch but by refining the previous r-division.

Lemma 2.4. *Given an r-division and $r' > r$, we can compute an r'-division in $n \lg(r'/r) + O(n\sqrt{\lg(r'/r)} + n \lg\lg r + r' \lg r')$ expected number of comparisons.*

Proof. Given an r-division corresponding to a random sample $R \subset S$, we (i) take a random sample $R' \subset S$ containing R of size r', and (ii) compute a point location structure for R' in $O(r' \lg r')$ number of comparisons. It remains to describe how to (iii) compute the conflict lists for R'.

We first identify the cell of $\mathrm{VD}(R)$ containing each corner point of S (if it exists). This can be done by a linear scan over all conflict lists for R, in time $O(\sum_{\Delta \in \mathrm{VD}(R)} |S_\Delta|)$. By Clarkson and Shor's analysis [17], this expression has expected value $O(r \cdot n/r) = O(n)$.

Next, for each cell $\Delta \in \mathrm{VD}(R)$, we build Seidel and Adamy's point location structure for the xy-projection of $\mathrm{VD}(R')$ restricted inside Δ. This takes time $O(\sum_{\Delta \in \mathrm{VD}(R)} |R'_\Delta| \lg |R'_\Delta|)$. By Clarkson's analysis [15], since R is a random sample of R', this expression has expected value $O(r \cdot (r'/r) \lg(r'/r)) = O(r \lg(r'/r))$.

For each corner point q of S lying inside a cell $\Delta \in \mathrm{VD}(R)$, we then locate the cell $\Delta' \in \mathrm{VD}(R')$ that contains q by Seidel and Adamy's query algorithm. This takes a total of $n \cdot (\lg \eta + O(\sqrt{\lg \eta}))$ comparisons, where $\eta = \max_{\Delta \in \mathrm{VD}(R)} |R'_\Delta|$. By Clarkson's analysis [15], since R is a random sample of R', the expected value of η is $O((r'/r) \lg r)$. By Jensen's inequality, the expected value of $n \cdot (\lg \eta + O(\sqrt{\lg \eta}))$ is $n \cdot (\lg(r'/r) + O(\lg \lg r + \sqrt{\lg(r'/r)}))$.

For each $s \in S$, we can then find all cells of $\mathrm{VD}(R')$ intersecting s by walking in $\mathrm{VD}(R')$, starting from the cell containing the corner of s. This takes time linear in the sum of the degrees of these cells, by breadth- or depth-first search. Afterwards, we obtain the conflict lists $S_{\Delta'}$ for all cells $\Delta' \in \mathrm{VD}(R')$. The cost is $O(\sum_{\Delta' \in \mathrm{VD}(R')} \delta_{\Delta'} |S_{\Delta'}|)$. By Clarkson and Shor's analysis [17], this expression has expected value $O(r' \cdot n/r') = O(n)$. □

Algorithm outline.

1. for $i = 1, 2, \ldots$ until all maxima are found:
2. compute an r_i-division from the previous r_{i-1}-division by Lemma 2.2
3. try to compute all maxima of S using the r_i-division by Lemma 2.1, if $h \lg^2 h \le r_i$

Analysis. We choose the sequence $r_i = \min\{2^{i^3}, n/\lg n\}$. Let ℓ satisfy $r_{\ell-1} < h \lg^2 h \le r_\ell$; then $\ell < \lg^{1/3}(h \lg^2 h) + 1$.

Line 2 takes at most $n \lg(r_i/r_{i-1}) + O(n\sqrt{\lg(r_i/r_{i-1})} + n \lg \lg r_{i-1}) = n(i^3 - (i-1)^3) + O(ni)$ comparisons. By a telescoping sum and an arithmetic series, the total over all iterations is at most $n\ell^3 + O(n\ell^2) = n \lg h + O(n \lg^{2/3} h)$.

Line 3 takes $O(n)$ comparisons, because we can declare the trial a failure as soon as the algorithm has discovered enough maximal points to violate $h \lg^2 h \le r_i$. The total over all iterations is $O(n\ell) = O(n \lg^{1/3} h)$.

Note that if $h \lg^2 h > n/\lg n$, we can compute all maxima from the $(n/\lg n)$-division in additional $O\left(\sum_{\Delta \in \mathrm{VD}(R)} |S_\Delta| \lg |S_\Delta|\right)$ comparisons. By Clarkson and Shor's analysis, with $r = n/\lg n$, this expression has expected value $O(r \cdot (n/r) \lg(n/r)) = O(n \lg \lg h)$.

We conclude that the overall number of comparisons is $n \lg h + O(n \lg^{2/3} h)$.

Remarks. The algorithm in Section 2.3 can be adapted to solve the 3D halfspace intersection problem (dual to 3D convex hull), if the definition of a "comparison" is extended to include more complicated predicates. However, the output-sensitive algorithm in this section does not seem to generalize (the problem lies in Lemma 2.4 and the lack of a feature analogous to "corners" for halfspaces).

2.5 Orthogonal Line Segment Intersection

In this subsection, we consider the problem of detecting an intersection between a set of n_v vertical segments and a set of n_h horizontal segments. Let $n = n_v + n_h$. The trivial algorithm uses $3n \lg n + O(n)$ comparisons: we just sort all coordinates (each segment is defined by 3 coordinate values); afterwards, no more comparisons are required.

A standard algorithm based on a vertical sweep line [4, 42] uses $(3n - n_v) \lg n + O(n)$ comparisons: we sort all coordinates except for the top y-coordinates of the vertical segments; then during the sweep, we only need to compare the top y-coordinate of each vertical segment with just one y-coordinate to detect intersections. Assuming that $n_v \geq n/2$ without loss of generality, the number of comparisons is thus $2.5n \lg n + O(n)$.

Other known $O(n \lg n)$-time algorithms (e.g., randomized incremental construction [17], hereditary segment tree [13], ...) do not seem to improve the constant factor.

We observe that the random sampling approach from Section 2.3 leads to a better result. The algorithm follows the exact same algorithm outline in Section 2.3, where now the vertical decomposition $\mathrm{VD}(S)$ is defined differently (by shooting an upward and a downward ray from each segment endpoint). In line 4, we do point location for just one of the two endpoints of the segment s. Using Seidel and Adamy's point location structure [43], we obtain a randomized algorithm that uses $n \lg n + O(n\sqrt{\lg n})$ expected number of comparisons.

Remarks. By the same approach, we can compute the vertical decomposition of a set of n horizontal segments in $n \lg n + O(n\sqrt{\lg n})$ expected number of comparisons. (The trivial algorithm requires $2n \lg n + O(n)$ comparisons.)

The same result for the detection and the vertical decomposition problem also holds for general nonorthogonal line segments, where a "comparison" now means either comparing two endpoints' x-coordinates, or testing whether an endpoint is above a segment.

For the problem of reporting all k intersections, a similar approach yields a randomized algorithm with $n \lg n + O(n\sqrt{\lg n}) + O(k)$ expected number of comparisons (with a further extended definition of "comparisons"). In the orthogonal case, the $O(k)$ term can be lowered to $O(n \lg(1 + k/n))$ with more effort.

3 Data Structure Problems

In this section, we study the number of comparisons needed for a fundamental class of geometric data structure problems: point location.

We begin with a trivial observation which, in the orthogonal setting, relate this class of problems to another well-studied topic in computational geometry—binary space partition (BSP) trees.

Formally, a *BSP tree* is a binary tree where each node is associated with a cell, and the cells of the two children of a node u is obtained by cutting u's cell into two by a hyperplane. The tree is *for* a given set S of objects if the interior of each leaf cell does not intersect the boundary of any object in S, and the cell of the root is the entire space. In an *orthogonal* BSP tree, all hyperplane cuts must be axis-parallel, and thus all cells must be axis-parallel boxes.

Observation 3.1. *For a set S of orthogonal polyhedral objects, there exists a data structure that can locate the object containing any query point in at most H comparisons iff there exists an orthogonal BSP tree for S with height at most H.*

Proof. The decision tree associated with the query algorithm is precisely an orthogonal BSP tree, and vice versa. □

(Note that the above equivalence concerns minimizing query complexity without space considerations, and does not address time–space tradeoffs.)

A large body of work has studied BSP tree constructions with good asymptotic worst-case bounds on the *size* of the tree [40, 23, 28, 44]. To get good query algorithms for point location, we need more, namely, BSP tree constructions with good *height* bounds; this direction has not been as well-pursued. Still, lower bounds on the query complexity of point location can be immediately derived from lower bounds on the sizes of BSP trees, since the height is at least lg of the size.

For example, for n disjoint boxes in 3D, a classic result by Paterson and Yao [40] (see also [23]) yielded a matching upper and lower bound of $\Theta(n^{3/2})$ on the worst-case size of an optimal orthogonal BSP tree. For another example, for a box subdivision with n boxes in 3D (interior-disjoint boxes that fill the entire space), a result by Hershberger, Suri, and Tóth [28] yielded matching upper and lower bound of $\Theta(n^{4/3})$. These size lower bounds immediately implies lower bounds of $(3/2)\lg n - O(1)$ and $(4/3)\lg n - O(1)$ on the worst-case number of comparisons needed for point location for disjoint boxes and box subdivisions in 3D respectively. Both upper bound constructions do not produce balanced BSP trees with good height bounds (one reason is the arbitrary occurrences of so-called "free cuts"). The main results of this section are new BSP tree constructions with height upper bounds matching these lower bounds up to lower-order terms in 3D.

In a higher constant dimension d, Dumitrescu, Mitchell, and Sharir [23] extended Paterson and Yao's result and obtained an $O(n^{d/2})$ upper bound on the BSP tree size for disjoint boxes, while Hershberger, Suri, and Tóth [28] obtained an $O(n^{(d+1)/3})$ upper bound for box subdivisions. We obtain new height upper bounds in higher dimensions as well. Unfortunately, optimality is not known in higher dimensions: for box subdivisions, the best lower bound by Hershberger et al. on the BSP tree size is $\Omega(n^{\beta(d)})$ for some function $\beta(d)$ that does not grow to infinity as d increases, but converges to a constant, the golden ratio $\phi = 1.618\cdots$.

3.1 A Weighted Search Technique

A key technique underlying all our data structures is the use of *weighted search trees*, inspired by Seidel and Adamy's data structure for planar point location [43] (which in turn is related to a classical point location method by Preparata [41]).

We present a lemma that encapsulates the essence of this idea. It may look simple in hindsight, but it provides a useful viewpoint—it allows us to rethink the problem of constructing a balanced BSP tree in terms of constructing a *multiway* space partition ("MSP") tree, with possibly large degree but very shallow (sublogarithmic!) height.

Formally, we define an *MSP tree* to be a tree where each node is associated with a cell, and the cells of the children of a degree-k node u is obtained by cutting u's cell using $k-1$ hyperplanes that do not intersect inside u's cell. In an *orthogonal* MSP tree, all hyperplane cuts must be axis-parallel, and thus the hyperplane cuts at each node must be orthogonal to the same direction.

Lemma 3.2. *Given an MSP tree with W leaves and height H, we can convert it to a BSP tree with the same W leaf cells and height* $\lg W + O(H)$.

Proof. For each node u in the MSP tree, let $W(u)$ denote the number of leaves underneath u. Let $v_1, \ldots, v_k$ be the children of u (in the order corresponding to the hyperplane cuts); then $W(u) = W(v_1) + \cdots + W(v_k)$. Replace the outgoing edges of u with a binary tree with root u and leaves $v_1, \ldots, v_k$ in the given order, so that the path length from u to v_i is bounded by $\lg W(u) - \lg W(v_i) + O(1)$. The existence of such a binary tree is well known [37].

Then a path $u_0, u_1, \ldots, u_j$ ($j \le H$) in the MSP tree is transformed into a path of length bounded by the telescoping sum $\sum_{i=0}^{j-1} [\lg W(u_i) - \lg W(u_{i+1}) + O(1)] \le \lg W + O(j)$. □

3.2 Point Location in 2D: Rederiving Seidel and Adamy's Method

Lemma 3.2 allows us to rederive Seidel and Adamy's planar point location result cleanly (at least the first version with $\lg n + O(\sqrt{\lg n})$ query time which ignores constants in the lower-order term). This follows from a simple MSP tree construction for n disjoint line segments with size $n2^{O(\sqrt{\lg n})}$ and height $O(\sqrt{\lg n})$.

Algorithm outline. We are given a set S of n disjoint line segments and a trapezoidal cell Δ that has two vertical sides and has its top and bottom sides defined by two segments of S. We construct an MSP tree inside Δ as follows:

1. divide Δ by r vertical lines so that each subcell has $O(n/r)$ endpoints
2. divide each subcell into subsubcells by the segments from S that completely cut across the subcell (these are "free cuts")
3. recursively build an MSP tree inside each subsubcell

Analysis. There are at most nr subsubcells. Let n_i be the number of segments intersecting the i-th subsubcell. Then $\max_i n_i = O(n/r)$, since each segment intersecting a subsubcell contributes to a vertex inside the subsubcell after the free cuts. Furthermore, $\sum_i n_i \le 2n$, since the two endpoints of a segment lie in two subsubcells.

Line 1 adds 1 to the height. Line 2 adds 1 as well.

The recurrences for the number of leaves $W(n)$ and the height $H(n)$ of the MSP tree are then given by

$$W(n) \le \sum_i W(n_i) \qquad \text{and} \qquad H(n) \le \max_i H(n_i) + 2$$

for some n_i's with at most nr terms, such that $\max_i n_i = O(n/r)$ and $\sum_i n_i \le 2n$.

For r fixed, the recurrences solve to $W(n) \le rn2^{O(\lg n/\lg r)}$ and $H(n) = O(\lg n/\lg r)$. Setting $r = 2^{\sqrt{\lg n}}$ gives $W(n) \le n2^{O(\sqrt{\lg n})}$ and $H(n) = O(\sqrt{\lg n})$. By Lemma 3.2, we obtain a BSP tree with height $\lg n + O(\sqrt{\lg n})$. This translates to a data structure that can answer point location queries in $\lg n + O(\sqrt{\lg n})$ comparisons, where in the nonorthogonal setting, a comparison is either comparing two endpoints' x-coordinates, or testing whether an endpoint is above a segment.

3.3 Point Location for Disjoint Boxes in $d \ge 3$ Dimensions

Point location queries for n disjoint boxes in a constant dimension $d \ge 3$ can be trivially answered in $d \lg n$ comparisons after sorting. We now present an improved result by obtaining a new height upper bound for BSP trees, using Lemma 3.2. We adopt the following known partitioning scheme:

Lemma 3.3. *Given n axis-parallel flats in a space of constant dimension d, and given r, we can divide space into $O(r^d)$ subcells so that the number of j-flats intersecting each subcell is $O(n/r^{d-j})$. This division is in fact an orthogonal MSP tree with height $O(1)$.*

Proof. (Sketch) A simple construction was given by Overmars and Yap [39] for the $j = d-2$ case, and was later rediscovered/generalized by Dumitrescu, Mitchell, and Sharir [23]. The lemma in the above form was also stated as Lemma 4.6 in [7], except for the last sentence. Upon examination of the proof, we see that this grid-like construction consists of d rounds where the i-th round makes hyperplane cuts orthogonal to the i-th axis; thus, it forms an MSP tree with height d. More precisely, here is a brief redescription of the proof presented in [7]:

Vertically project the input onto the first $d-1$ dimensions and construct the partition by induction. Then lift each cell to get a vertical column γ along the d-th dimension. Partition γ with $O(r)$ cuts using hyperplanes orthogonal to the d-th axis, so that the number of i-flats orthogonal to the d-th axis and intersecting each subcell is a factor of r less than that for γ, for each $i \in \{0, \ldots, d-1\}$. Clearly, the total number of subcells in this construction is $O(r^d)$, and the total number of rounds of hyperplane cuts is indeed d.

Let $n_i^{(d)}$ be the maximum number of i-flats intersecting each subcell in this d-dimensional construction. If an i-flat f is not orthogonal to the d-th axis, the vertical projection of f is an $(i-1)$-flat. If f is orthogonal to the d-th axis, the vertical projection of f is an i-flat. We therefore have

$$n_i^{(d)} \le n_{i-1}^{(d-1)} + \frac{n_i^{(d-1)}}{r}.$$

With the trivial base cases, it follows by induction that $n_i^{(d)} = O(n/r^{d-i})$. □

Algorithm outline. Given a cell Δ, we construct an orthogonal MSP tree inside Δ as follows:

1. divide Δ into subcells by applying Lemma 3.3 to the flats through the faces of S intersecting Δ
2. divide each subcell into subsubcells by the $(d-1)$-faces of S that completely cut across the subcell (these are "free cuts")
3. recursively build an orthogonal MSP tree inside each subsubcell

Analysis. The number of subsubcells is trivially bounded by $nr^{O(1)}$. Let n_i be the number of boxes intersecting the i-th subsubcell. Then $\max_i n_i = O(n/r^2)$, since the number of $(d-2)$-faces intersecting each subcell is $O(n/r^2)$ by Lemma 3.3, and each box intersecting a subsubcell contributes to a $(d-2)$-face intersecting the subsubcell after the free cuts. Furthermore, $\sum_i n_i = O(r^d \cdot n/r^2)$, since each box can intersect at most one subsubcell of each subcell, and there are $O(r^d)$ subcells.

Line 1 adds $O(1)$ to the height. Line 2 adds just 1 to the height, because the free cuts of each subcell must be orthogonal to the same direction.

The recurrences for the number of leaves $W(n)$ and the height $H(n)$ of the MSP tree are then given by

$$W(n) \le \sum_i W(n_i) \qquad \text{and} \qquad H(n) \le \max_i H(n_i) + O(1)$$

for some n_i's with at most $nr^{O(1)}$ terms, such that $\max_i n_i = O(n/r^2)$ and $\sum_i n_i = O(r^d \cdot n/r^2)$.

We choose $r = n^\varepsilon$ for a sufficiently small constant $\varepsilon > 0$. By induction, it is straightforward to verify that the recurrences solve to $W(n) = O(n^{d/2} \lg^{O(1)} n)$ and $H(n) = O(\lg \lg n)$. By Lemma 3.2,

we obtain an orthogonal BSP tree with height $(d/2)\lg n + O(\lg\lg n)$ and thus a data structure that can answer point location queries in $(d/2)\lg n + O(\lg\lg n)$ comparisons.

3.4 Point Location for Box Subdivisions in $d \geq 3$ Dimensions

For the case of disjoint boxes that are space-filling, i.e., that form a spatial subdivision, we can improve the upper bound further. The extra ingredient is provided by a subroutine of Hershberger, Suri, and Tóth [28]:

Lemma 3.4. *In the special case of a box subdivision S inside a cell Δ where no $(d-3)$-faces intersect the interior of Δ, there is an orthogonal MSP tree for S with $n2^{O(\sqrt{\lg n})}$ leaves and height $O(\sqrt{\lg n})$.*

Proof. (Sketch) This special case was addressed in Lemma 3.4 (for $d = 3$) and Lemma 5.1 (for general d) in Hershberger, Suri, and Tóth's paper [28]. They only stated the existence of a BSP tree with size $O(n)$. Specifically, they proved the existence of a series of free cuts after which the subproblems become two-dimensional (i.e., the subdivisions are liftings of 2D subdivisions). For these 2D subdivisions, we can switch to Seidel and Adamy's method (in Section 3.2) to get MSP subtrees of total size $n2^{O(\sqrt{\lg n})}$ and height $O(\sqrt{\lg n})$. Upon close examination of their proof, we see that the series of free cuts are all orthogonal to a common direction; thus, they add just 1 to the height of the MSP tree. More precisely, here is a brief reinterpretation of their proof (the original proof uses induction, which we find it clearer to avoid):

Let $\Delta = [0,1]^d$. We are given a box subdivision where no $(d-3)$-faces intersect the interior of Δ. Thus, each box can be written in the form $\{a_i \leq x_i \leq b_i, a_j \leq x_j \leq b_j\}$ for some a_i, b_i, a_j, b_j; we say that the box is of *type* ij. (For boxes of the form $\{a_i \leq x_i \leq b_i\}$, we can arbitrarily pick an index $j \neq i$ and set $a_j = 0, b_j = 1$.)

The types of all the given boxes can be viewed as edges of a graph G, with the indices $\{1, \ldots, d\}$ as vertices.

We claim that every two edges in G must share a common vertex: If this were not true, there would be two boxes of types ij and $i'j'$ for distinct indices i, j, i', j'. The interiors of the two boxes contain two $(d-2)$-flats $\{x_i = c_i, x_j = c_j\}$ and $\{x_{i'} = c_{i'}, x_{j'} = c_{j'}\}$ for some $c_i, c_j, c_{i'}, c_{j'} \in (0,1)$. But then the pair would intersect at $x_i = c_i, x_j = c_j, x_{i'} = c_{i'}, x_{j'} = c_{j'}$.

It is not difficult to see that the only possible graphs satisfying the property that every two edges are adjacent are (i) a triangle, and (ii) a star.

Suppose that G is a triangle, say, 123. We show that this is not possible. Consider three boxes of types 12, 23, and 31. The interiors of these three boxes contain three $(d-2)$-flats $\{x_1 = c_1, x_2 = d_2\}$, $\{x_2 = c_2, x_3 = d_3\}$, and $\{x_3 = c_3, x_1 = d_1\}$ for some $c_1, c_2, c_3, d_1, d_2, d_3 \in (0,1)$. Because the boxes are space-filling, a point at $x_1 = c_1, x_2 = c_2, x_3 = c_3$ must be covered by some box. By symmetry, we may assume that this box is of type 12; it contains the $(d-2)$-flat $\{x_1 = c_1, x_2 = c_2\}$. But then $\{x_1 = c_1, x_2 = c_2\}$ and $\{x_2 = c_2, x_3 = d_3\}$ would intersect at $x_1 = c_1, x_2 = c_2, x_3 = d_3$.

From now on, suppose that G is a star, i.e., all edges are incident to a common vertex, say, 1. Sweep space using a hyperplane $h = \{x_1 = t\}$ for some value t representing time.

We claim that at any time, h can hit the interiors of boxes of only one type: If this were not true, h would hit the interiors of two boxes of type $1j$ and $1j'$ for some $j \neq j'$. The interior of these boxes contain two $(d-2)$-flats $\{x_1 = t, x_j = c_j\}$ and $\{x_1 = t, x_{j'} = c_{j'}\}$ for some $c_j, c_{j'} \in (0,1)$. But then the pair would intersect at $x_1 = t, x_j = c_j, x_{j'} = c_{j'}$.

During the sweep, we will make a hyperplane cut at h whenever the box type hit by h changes. (These are "free cuts".) Then all these hyperplane cuts are indeed orthogonal to a common direction, namely x_1. Furthermore, between two hyperplane cuts, there are boxes of only one type $1j$ for some j, and these are liftings of a 2D subdivision along the x_1 and x_j axes. □

Algorithm outline. We follow the same algorithm outline as in Section 3.3, except for an extra base case: when no $(d-3)$-faces intersect the interior of a cell Δ, we build an orthogonal MSP tree directly by Lemma 3.4.

Analysis. As before, the number of subsubcells is trivially bounded by $nr^{O(1)}$. Let n_i be the number of boxes intersecting the i-th subsubcell. As before, $\sum_i n_i = O(r^d \cdot n/r^2)$.

Let m be the number of $(d-3)$-faces intersecting the interior of Δ, and m_i be the number of $(d-3)$-faces intersecting the interior of the i-th subsubcell. A direct application of Lemma 3.3 implies that $\max_i m_i = O(n/r^3)$, but if we duplicate each $(d-3)$-flat $\lceil n/m \rceil$ times first before applying Lemma 3.3, the number of flats remains $O(n)$ and we get $\max_i m_i = O((n/r^3)/(n/m)) = O(m/r^3)$.

The recurrences for the number of leaves and the height of the MSP tree become

$$W(n,m) \le \sum_i W(n_i, m_i) \qquad \text{and} \qquad H(n,m) \le \max_i H(n_i, m_i) + O(1)$$

for some n_i's and m_i's with at most $nr^{O(1)}$ terms, such that $\max_i m_i = O(m/r^3)$ and $\sum_i n_i = O(r^d \cdot n/r^2) = O(nr^{d-2})$.

Lemma 3.4 provides the base case $W(n,0) \le n2^{O(\sqrt{\lg n})}$ and $H(n,0) = O(\sqrt{\lg n})$.

We choose $r = m^\varepsilon$. By induction, it is straightforward to verify that the recurrences solve to $W(n,m) \le nm^{(d-2)/3}2^{O(\sqrt{\lg n})}$ and $H(n,m) = O(\lg\lg m + \sqrt{\lg n})$. So, $W(n,n) \le n^{(d+1)/3}2^{O(\sqrt{\lg n})}$ and $H(n,n) = O(\sqrt{\lg n})$. By Lemma 3.2, we obtain an orthogonal BSP tree with height $((d+1)/3)\lg n + O(\sqrt{\lg n})$ and thus a data structure that can answer point location queries in $((d+1)/3)\lg n + O(\sqrt{\lg n})$ comparisons.

Remarks. The space usage of our data structures is very high: $O(n^{d/2}\log^{O(1)} n)$ for disjoint boxes and $O(n^{(d+1)/3}2^{O(\sqrt{\lg n})})$ for box subdivisions. In the further special case of a box subdivision that is a BSP (possibly unbalanced), we can adopt an approach from [8, Section 4.2] of using tree separators, to reduce space to $O(n)$ while preserving the query time bound.

4 Open Problems

Our results open up a plethora of interesting questions:

- Can the d-dimensional maxima problem be solved using better than $dn\log n$ comparisons for $d \ge 4$? (Note that here, running time may not correlated with the number of comparisons, as currently the best algorithm for d-dimensional maxima has running time $O(n\log^{d-3} n)$ [9].)
- What is the complexity of the counting version of the orthogonal line segment intersection problem?

- What is the best deterministic bound on the number of comparisons for 3D maxima or 2D orthogonal segment intersection detection?
- Can one prove optimality of the lower-order terms for any of our results?
- Is the $O(\sqrt{\log n})$ lower-order term in Seidel and Adamy's result optimal for 2D orthogonal point location? (Their lower bound was proved only for a restricted setting in which all horizontal cuts are made by input segments completely crossing a cell.)
- Can one prove a lower bound for point location in d-dimensional disjoint boxes with a constant factor that converges to infinity as a function of d?

There are countless other geometric problems one could study regarding constant factors. For example, a challenging one would be triangulations of orthogonal polygons.

References

[1] P. Afshani, J. Barbay, and T. M. Chan. Instance-optimal geometric algorithms. In *Proc. 50th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 129–138, 2009.

[2] S. Arya, T. Malamatos, D. M. Mount, and K. C. Wong. Optimal expected-case planar point location. *SIAM J. Comput.*, 37:584–610, 2007.

[3] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 80–86, 1983.

[4] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, 1979.

[5] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete Comput. Geom.*, 16:361–368, 1996.

[6] T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete Comput. Geom.*, 16:369–387, 1996.

[7] T. M. Chan. Klee's measure problem made easy. In *Proc. 54th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 410–419, 2013.

[8] T. M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. *ACM Trans. Algorithms*, 9(3):22, 2013.

[9] T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proc. 27th Annu. ACM Sympos. Comput. Geom.*, pages 1–10, 2011.

[10] T. M. Chan and M. Pătraşcu. Transdichotomous results in computational geometry, I: Point location in sublogarithmic time. *SIAM J. Comput.*, 39:703–729, 2009.

[11] T. M. Chan, J. Snoeyink, and C. K. Yap. Primal dividing and dual pruning: Output-sensitive construction of 4-d polytopes and 3-d Voronoi diagrams. *Discrete Comput. Geom.*, 18:433–454, 1997.

[12] B. Chazelle and D. P. Dobkin. Intersection of convex objects in two and three dimensions. *J. ACM*, 34(1):1–27, January 1987.

[13] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994.

[14] B. Chazelle and J. Matoušek. Derandomizing an output-sensitive convex hull algorithm in three dimensions. *Comput. Geom. Theory Appl.*, 5:27–32, 1995.

[15] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete Comput. Geom.*, 2:195–222, 1987.

[16] K. L. Clarkson. More output-sensitive geometric algorithms. In *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 695–702, 1994.

[17] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[18] W. Cunto and J. I. Munro. Average case selection. *J. ACM*, 36:270–279, 1989.

[19] D. P. Dobkin and R. J. Lipton. On the complexity of computations under varying sets of primitives. *J. Comput. Syst. Sci.*, 18:86–91, 1979.

[20] D. Dor and U. Zwick. Selecting the median. *SIAM J. Comput.*, 28:1722–1758, 1999.

[21] D. Dor and U. Zwick. Median selection requires $(2 + \epsilon)n$ comparisons. *SIAM J. Discrete Math.*, 14:312–325, 2001.

[22] T. Dubé. Dominance range-query: The one-reporting case. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 208–217, 1993.

[23] A. Dumitrescu, J. S. B. Mitchell, and M. Sharir. Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles. *Discrete Comput. Geom.*, 31:207–227, 2004.

[24] J. Ford, R. Lester, and S. M. Johnson. A tournament problem. *Amer. Math. Monthly*, 66:387–389, 1959.

[25] G. N. Frederickson and D. B. Johnson. The complexity of selection and ranking in $X + Y$ and matrices with sorted rows and columns. *J. Comput. Syst. Sci.*, 24:197–208, 1982.

[26] M. J. Golin. A provably fast linear-expected-time maxima-finding algorithm. *Algorithmica*, 11:501–524, 1994.

[27] G. H. Gonnet and J. I. Munro. Heaps on heaps. *SIAM J. Comput.*, 15:964–971, 1986.

[28] J. Hershberger, S. Suri, and C. D. Tóth. Binary space partitions of orthogonal subdivisions. *SIAM J. Comput.*, 34:1380–1397, 2005.

[29] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Inform. Process. Lett.*, 2:18–21, 1973.

[30] K. Kaligosi, K. Mehlhorn, J. I. Munro, and P. Sanders. Towards optimal multiple selection. In *Proc. 32nd Int. Colloq. Automata, Languages and Programming*, volume 3580 of *Lecture Notes Comput. Sci.*, pages 103–114. Springer-Verlag, 2005.

[31] D. G. Kirkpatrick and R. Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 89–96, 1985.

[32] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15:287–299, 1986.

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

[34] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22:469–476, 1975.

[35] Z. Li and B. A. Reed. Heap building bounds. In *Proc. 9th Workshop Algorithms Data Struct.*, volume 3608, pages 14–23. Springer-Verlag, 2005.

[36] Giuseppe Liotta, Franco P. Preparata, and Roberto Tamassia. Robust proximity queries: An illustration of degree-driven algorithm design. *SIAM J. Comput.*, 28(3):864–889, 1998.

[37] K. Mehlhorn. *Data Structures and Algorithm 1: Sorting and Searching*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin/Heidelberg, Germany, 1981.

[38] T. Ottmann, S. Schuierer, and S. Soundaralakshmi. Enumerating extreme points in higher dimensions. In *Proc. 12th Sympos. Theoret. Aspects Comput. Sci.*, volume 900 of *Lecture Notes Comput. Sci.*, pages 562–570. Springer-Verlag, 1995.

[39] M. H. Overmars and C.-K. Yap. New upper bounds in Klee's measure problem. *SIAM J. Comput.*, 20:1034–1045, 1991.

[40] M. S. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. *J. Algorithms*, 13:99–113, 1992.

[41] F. P. Preparata. A new approach to planar point location. *SIAM J. Comput.*, 10(3):473–482, 1981.

[42] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[43] R. Seidel and U. Adamy. On the exact worst case complexity of planar point location. *J. Algorithms*, 37:189–217, 2000.

[44] C. D. Tóth. Binary space partitions: Recent developments. In *Combinatorial and Computational Geometry*, volume 52 of *MSRI Publications*, pages 525–552. Cambridge University Press, 2005.

[45] P. van Emde Boas. On the $\Omega(n \log n)$ lower-bound for convex hull and maximal vector determination. *Inform. Process. Lett.*, 10:132–136, 1980.

A Appendix: Lower Bounds

Kirkpatrick and Seidel [32, 31] proved an $\Omega(n \lg h)$ lower bound for the 2D convex hull and 2D maxima problem (the version in which the output need not be in sorted order) in the algebraic computation tree model by using Ben-Or's technique [3]. Ben-Or's technique also implies an $\Omega(n \lg n)$ lower bound for the orthogonal line segment intersection detection problem. The technique has hidden constant factors, however.

In this section, we note that for 2D maxima and orthogonal segment intersection detection, the hidden constant factor in the lower bounds can be made equal to 1 in the *linear* decision tree model, where a comparison refers to testing the sign of a linear combination of the input values. Although the model is not as powerful as higher-degree algebraic decision trees, it permits ordinary comparisons between two input coordinates, which are what this paper focuses on anyway. For linear decision trees, a simpler technique [19, 45] suffices. The main observation is that cells in a linear decision tree are intersections of halfspaces and are thus convex.

Maxima in 2D. Let F consist of all functions $f : \{1, \ldots, n\} \to \{1, \ldots, h\}$ such that $f(1), \ldots, f(h)$ is a permutation of $\{1, \ldots, h\}$. Given $f \in F$, define S_f be the sequence of n points where the i-th point is $(f(i) - \varepsilon i, -f(i) - \varepsilon i)$. Then S_f has exactly h maximal points, namely the first h points.

Suppose that the two inputs S_f and S_g ($f, g \in F$) end up in the same leaf cell of the decision tree. Imagine moving the point positions from S_f to S_g linearly. At any moment in time, the input point sequence belongs to the same leaf cell, by convexity. The i-th point can only move within the line $\{(t - \varepsilon i, -t - \varepsilon i) : t \in \mathbf{R}\}$.

For $i, j \in \{1, \ldots, h\}$, the i-th and j-th point cannot change order, for otherwise one of these two points would not be maximal at some moment in time. Thus, $f(1) = g(1), \ldots, f(h) = g(h)$. In particular, the staircase of the first h points stays unmoved.

For $i \in \{h+1, \ldots, n\}$, the i-th point cannnot cross this staircase, for otherwise the set of maximal points would change (assuming a sufficiently small $\varepsilon > 0$). Thus, $f(i) = g(i)$.

We conclude that the inputs S_f over all $f \in F$ must lie in distinct leaf cells. Consequently, the number of leaf cells is at least $|F| = h!h^{n-h}$, and the height of the decision tree is at least $\lg |F| = n \lg h - O(h)$.

In fact, the average depth over $|F|$ distinct leaves in the decision tree is at least $\lg |F|$. It follows that the expected number of comparisons on the input S_f for a random $f \in F$ is at least $n \lg h - O(h)$. By Yao's principle, this gives the same expected lower bound for randomized algorithms.

Orthogonal line segment intersection detection. Let F consist of all pairs $f = (f_v, f_h)$ where $f_v, f_h : \{1, \ldots n/2\} \to \{1, \ldots, n/2\}$ are bijections. Given $f \in F$, define S_f to be a sequence of $n/2$ vertical and $n/2$ horizontal line segments where the i-th vertical segment has endpoints $(f(i), -1)$ and $(f(i), 1)$ and the i-th horizontal segment has endpoints $(f(i)+1/3, 0)$ and $(f(i)+2/3, 0)$. Then S_f has no intersections.

Suppose that the two inputs S_f and S_g ($f, g \in F$) end up in the same leaf cell of the decision tree. Imagine moving the coordinates of the segments from S_f to S_g linearly. At any moment in time, the input point sequence belongs to the same leaf cell, by convexity. Each segment can only move horizontally. The segments cannot change x-order, for otherwise the input would have to change from no to yes at some moment in time. We thus have $f = g$.

We conclude that the inputs S_f over all $f \in F$ must lie in distinct leaf cells. Consequently, the number of leaf cells is at least $|F| = (n/2)!^2$, and the height of the decision tree is at least $\lg |F| = n \lg n - O(n)$.

As before, we also obtain the same expected lower bound for randomized algorithms.