PREPRINT. Computational Geometry: Theory and Applications, 8(3):115-122, 1997.

On the Difficulty of Range Searching

Arne Andersson[*] Kurt Swanson[*]
Dept. of Computer Science, Lund University,
Box 118, S-221 00 LUND, Sweden

Abstract

We consider the general problem of (2-dimensional) range reporting allowing arbitrarily convex queries. We show that using a traditional approach, even when incorporating techniques like those used in fusion trees, a polylogarithmic query time can not be achieved unless more than linear space is used. Our arguments are based on a new non-trivial lower bound in a model of computation which, in contrast to the pointer machine model, allows for the use of arrays and bit manipulation. The crucial property of our model, *Layered Partitions*, is that it can be used to describe all known algorithms for processing range queries, as well as many other data structures used to represent multi-dimensional data. We show that $\Omega\left(\frac{\log n}{\log T(n)}\right)$ partitions must be used to allow queries in $O(T(n)+k)$ time, for n total and k reported elements, and for any growing function $T(n)$.

1 Introduction

1.1 Discrepancy between upper and lower bound

The complexity of range searching has been a long standing open question. The problem is fundamental and easy to formulate: given a set of points in a multidimensional space, create a data structure that facilitates reporting of all points inside a given query region. This formulation is frequently called *range reporting*. Although many attempts have been made, the time-space tradeoff for this problem is still unclear.

This problem has been studied in several different computational models. Taking a general approach, we note the following trivial lower bounds: Let $F(n)$ be the time required to find *one* point in the query region—or to discover that the region is empty—in a set of n points. Then, an immediate lower bound on time complexity is $\Omega(F(n)+k)$, where k is the number of points to be reported. The same lower bound applies when $F(n)$ is the cost

[*]email: {arne, kurt}@dna.lth.se

of counting the number of points in the region or computing the weighted sum of a set of weighted points.

A strong model of computation is the cell probe model. By a modification of the proof by Miltersen [15], a lower bound of $\Omega(\log^{1/3-o(1)} n)$ can be obtained for $F(n)$, the time required to find one point in the query region [14]. (The original bound given in that article is expressed in terms of the size of the universe. This is also the case for similar lower bounds on *existential range searching*, which reports whether or not a given region is empty, given by Miltersen et al [16].)

In the arithmetic model of computation, only the number of arithmetic operations needed to answer a query are counted. The lower bound by Chazelle [9] for the problem of *dominance searching* (where the sum of weights of all points dominated by a given point is computed), applies to our problem.

Lower bounds in the models mentioned above are attractive because of the strengths of the models. However it has shown to be difficult to provide tight bounds in these models. In a weaker model of computation, the pointer machine model, there exist tight bounds. Here $F(n) = \Omega(\log n)$ even in the one-dimensional case. In this model, it has been shown [8] that $\Theta(n \log n / \log \log n)$ space is necessary and sufficient in order to achieve optimal query time complexity. In fact, not even polylogarithmic time can be achieved with less space.

However, the restrictions of the pointer machine model are not realistic. This weakness is particularly evident for range reporting since it has been explicitly demonstrated that the use of arrays and bit manipulation can help reduce asymptotic complexity. In such a more general, and more realistic, model we can expect faster searching than $\Omega(\log n)$ [1, 2, 11]. As another possible improvement, we may reduce space requirements by packing information about more than one point into one machine word. This option of storing point sets in compressed form has been utilized by Chazelle [7] in the special case of *rectilinear* range queries; a query cost of $O(\log^4 n + k)$ can be achieved using only $O(n)$ space, and a query cost of $O(\log n + k)$ can be achieved using $O(n \log^{\epsilon} n)$ space on the RAM model.

But what about the general problem of non-rectilinear queries?

1.2 A lower bound in a relevant model

We concentrate on two-dimensional range reporting with convex query regions. It should be noted that our lower bound only holds for a (relevant) class of algorithms to solve range reporting, and not the problem itself.

In examining the large set of known data structures, one finds that they all have one property in common: they may all be viewed as representing one or more partitions of the plane, where each partition divides the plane into $\Theta(n)$ convex areas. When a query is made, the answer is given by intersecting the query region with a selected subset of these partitions.

Based on this observation, we define the model of *Layered Partitions*, which encompasses all known solutions to the range searching problem, as well as many other data structures used in computational geometry. Our lower bound on the time-space tradeoff is given in terms of the number of partitions needed in order to achieve a certain query cost. We show that, in order to support queries in time $O(T(n) + k)$, $\Omega(\log n / \log T(n))$ partitions must be represented. Hence, if each partition does require $\Theta(n)$ space (and there is no evidence pointing to the contrary for the general case of convex query regions), then any algorithm based on layered partitions, which supports range reporting in $O(\log^c n + k)$, $c = O(1)$, time requires $\Omega\left(\frac{n \log n}{\log \log n}\right)$ space.

Our model of computation is general in the sense that it allows all kinds of bit-manipulation techniques, such as those used in fusion trees [11], to be used in order to speed up queries, thus avoiding inherent weaknesses in the pointer machine model.

2 Computational model

Our computational model is based on a data structure paradigm which can be used to describe all data structures for the range reporting problem found in the literature, as well as many other data structures used to represent multi-dimensional data.

A central part of our model is the *partition*. Given a set of points in the plane, we define the "universe" as a rectangle containing all points in the set. A partition divides the universe into $\Theta(n)$ convex regions, where each region contains at most one point.

A data structure in the model represents a set of partitions $P_1, P_2, \ldots$ in the plane. Range queries are processed in the following way:

1. Split the query region into m subregions $U_1, \ldots, U_m$.
2. Associate each S_i with a partition $P(U_i)$.
3. Examine all regions in $P(U_i)$ that intersect U_i and report which points are contained in U_i.

The cost of processing the query is defined as:

$$\sum_{i=1}^{m} \text{number of regions in } P(U_i) \text{ that intersect } U_i$$

Note that we only consider the time required to access information in partitions, and not the time needed to determine how to divide the query region into subregions, nor the time to determine which partitions to use, nor time spent searching in any ancillary data structures. Thus differences between the RAM and decision tree models are negated.

We claim that this model covers the classical data structures used to solve this, and similar, problems. Among others, the following data structures can be described as layered partitions: k-d-Trees [17], Multistage direct access (multilevel k-ranges) [4], Filtering search [5], Range Trees [3] (see also [18] and [12]), Quad trees [10], Priority search trees [13], Voronoi diagrams, and M-structures [7] (see also [6]).

As an example, we indicate how to describe, in terms of layered partitions, priority search trees as well as the data structure used in filtering search. In Figure 1, we illustrate how to view a priority search tree as a layered partition (a single layer). In a priority search tree, the point with highest y-coordinate is stored in the root. The rest of the points are divided between the two subtrees according to a *split value* which is also stored in the root. All points whose x-coordinate is less than the split value are stored in the left subtree, the other points are stored in the right subtree. A priority search tree supports range queries where the query region is a rectangle for which the topmost edge in the rectangle is located above all stored points.

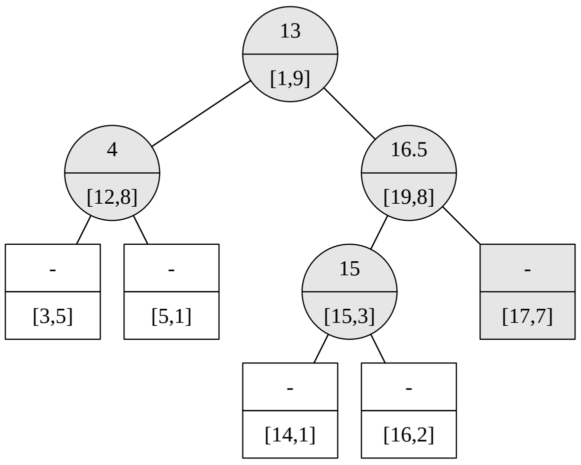
In the figure, the universe covers $[0,20] \times [0,10]$ and the query region (the shaded rectangle) is $[8,18] \times [6,10]$. In each tree node, the upper values are split values and the lower values are point coordinates. Split values are not needed for the leaves. The partition created by the tree is illustrated below the tree; each node corresponds to one region. Vertical segments represent split values, and horizontal segments separate each node's point from those of its subtree. The horizontal segments are somewhat arbitrarily drawn, but conform to the functional nature of priority search trees as well as layered partitions. This particular query rectangle contains five regions, the corresponding nodes are shaded.

In filtering search, the data structure is organized in $\Theta(\log n/\log\log n)$ layers. Each layer may be described as consisting of three partitions. At the ith layer, we have the following three partitions:

- The first partition divides the universe into $\Theta(\log^i n)$ vertical segments, each segment containing $\Theta(n/\log^i n)$ points. Each segment is in turn divided into horizontal segments, each segment containing one point.

- The second partition divides the universe into the same $\Theta(\log^i n)$ vertical segments. This time, however, each segment is represented as a priority search tree, oriented on the left edge of the segment. A priority search tree may in turn be described as a partition as previously shown.

- The third partition is similar to the second one, with the difference that the priority search trees are based on the right edge of their segments.



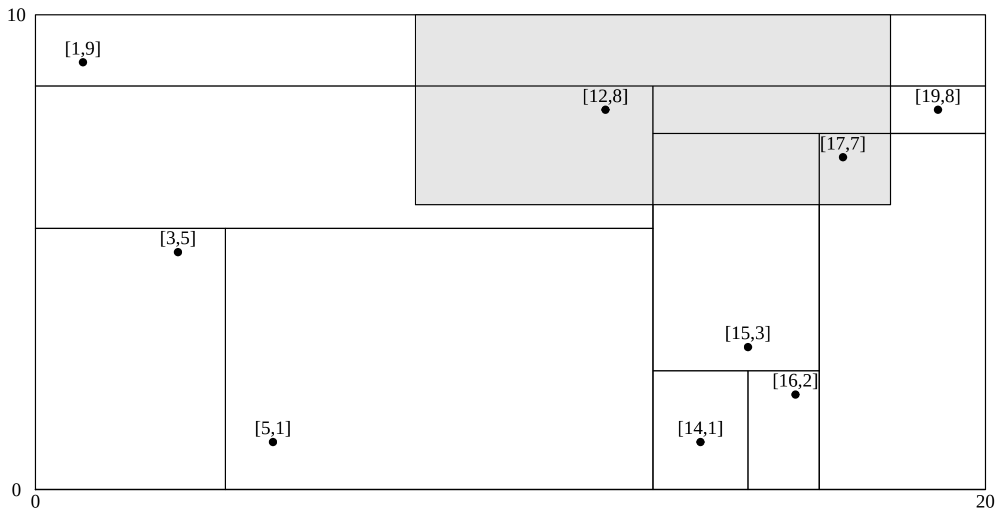


Figure 1: Searching in a priority search tree

3 Lower bound

Our lower bound is based upon a specific example. We describe a simple layout pattern for points in the plane, and prove that, according to the above described model of computation, any algorithm that solves range searching in $O(T(n) + k)$ time, must represent $\Omega\left(\frac{\log n}{\log T(n)}\right)$ partitions.

For the sake of simplicity, we shall describe this layout in terms of rectangular queries. However, our lower bound construction can easily be extended to a set of convex non-rectangular queries. We indicate how to perform this extension below.

We arrange the point set to create classes of empty rectangles R_1, R_2, R_3, ... Note that the rectangles are not explicitly part of the final point set—they are only for the construction. We denote the entire set of rectangles as $R = R_1 \cup R_2 \cup R_3 \cup ...$ The class R_i is constructed iteratively in the following way (see Figure 2):

1. Initially $i = 1$ and $R = R_1$ which contains one rectangle, and we are given values of n and $T(n)$.
2. Set $i = i + 1$.
3. Make $T^3(n)$ copies of the entire set of rectangles R and place them on a horizontal line. This implies that the number of rectangles of class $R_j, (\forall j \mid 1 \leq j < i)$ increases by a factor of $T^3(n)$.
4. Add a class of rectangles R_i as $T^{3(i-1)}(n)$ long rectangles, evenly spaced over the universe from top to bottom, each one intersecting all copies of one rectangle made in the previous step (as shown in Figure 2).
5. Make the holes (rectangular regions between intersecting rectangles) small enough so that their total area is negligible.
6. If there are enough points remaining for another iteration, go to 2.
7. Place one point in each hole between all the rectangles of R. If there are any remaining points, pack them arbitrarily in one of the holes.

We now indicate how this construction can be rephrased for a set of convex and non-rectangular queries. To do this, we observe that we have left small gaps between all rectangles. Hence, instead of placing our points on strict lines, we may arrange them in a slightly more irregular pattern, creating convex, but not necessarily rectangular, empty query regions.

Lemma 1 *The number of classes of rectangles that can be created by the above described process is* $\Omega\left(\frac{\log n}{\log T(n)}\right)$.

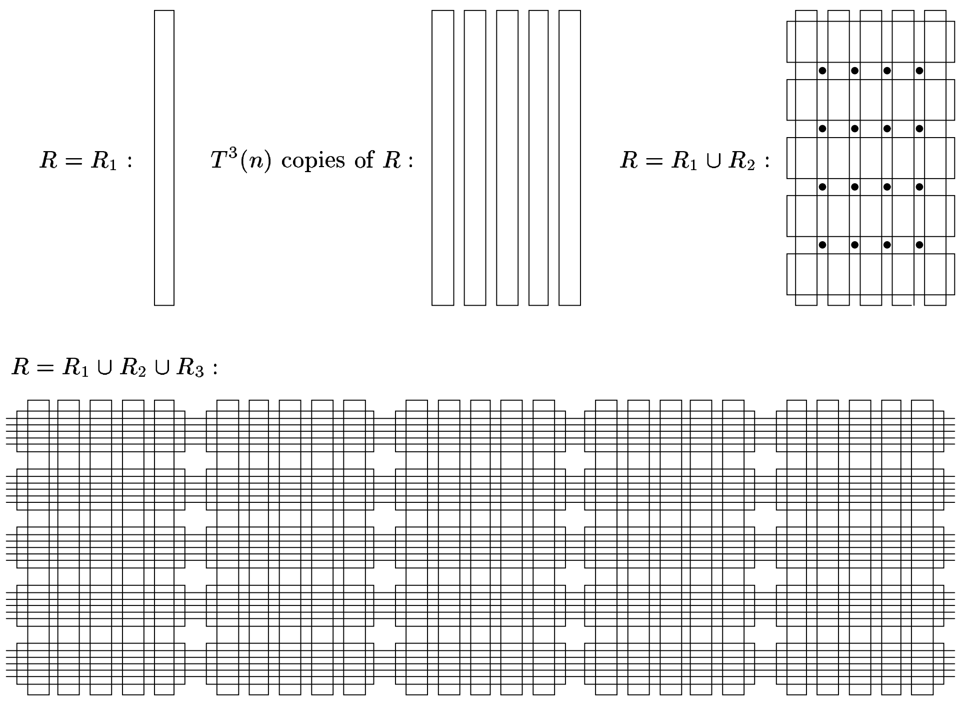


Figure 2: The growth of R, for $T^3(n) = 5$, yielding 16 holes for $R = R_1 \cup R_2$, and 176 total holes for $R = R_1 \cup R_2 \cup R_3$. (The 25 rectangles in R_3 are shown as lines. Only points in R_2 are drawn. For simplicity's sake, the unrealistic value of 5 for $T^3(n)$ has been used.)

Proof

We assume, without loss of generality, that the number of points, n, exactly fills the last class, without any remaining points. If this were not the case, at most one fewer class would be created having no effect on the asymptotic bound. Let $S(i)$ equal the number of points required to create i classes of rectangles. When creating the ith class, we create $T^3(n)$ copies of everything that we had before, the number of points needed for this purpose is $S(i-1)T^3(n)$. Next, we add $T^{3(i-1)}(n)$ rectangles. Each of these rectangles crosses $T^3(n)$ rectangles in R_{i-1}, creating $(T^{3(i-1)}(n)-1)(T^3(n)-1)$ new holes for level R_i, and we add one point in each hole, which gives us:

$$S(i) \quad = \quad S(i-1)T^3(n) + (T^{3(i-1)}(n)-1)(T^3(n)-1)$$

We first prove that $S(i) < T^{4i}(n)$ by induction. As $S(1) = 0$, then $S(2) = (T^{3\cdot 1}(n) - 1)(T^3(n) - 1) < T^{4\cdot 2}(n)$. By induction assume $S(i-1) < T^{4(i-1)}(n)$, then, for $T(n) > 1$ and $i > 2$:

$$\begin{aligned} S(i) &< T^{4(i-1)}(n)T^3(n) + (T^{3(i-1)}(n)-1)(T^3(n)-1) \\ &< T^{4i-1}(n) + T^{3i}(n) - T^{3(i-1)}(n) - T^3(n) + 1 < T^{4i}(n) \end{aligned}$$

We now set $n = S(i)$ and solve for i yielding $\Omega\left(\frac{\log n}{\log T(n)}\right)$. □

Next, we study how many partitions must be maintained in order to perform range queries efficiently. Intuitively, our goal is to show that there must be at least as many partitions as there are classes of empty rectangles.

Recall that in our model a partition consists of convex regions. We say that a region is *q-sized* if both its height and width is at least $1/T(n)$ times the height and width of a rectangle in R_q.

Lemma 2 *A q-sized region used in a partition is not s-sized if $q \neq s$.*

Proof The proof follows from the way our R_q's are constructed and the fact that no region used in a partition may contain more than one point.

Without loss of generality, assume $q < s$. The height of a q-sized region is greater than $1/T(n)$ times the height of a rectangle in R_q, by definition. The height of a q-sized region is greater than $T^2(n)$ times the height of a rectangle in R_s due to the construction of R_q and R_s.

Assume a q-sized region used in a partition is at least $1/T(n)$ times the width of a rectangle in R_s. Then the q-sized region must overlap at least $T^2(n)$ rectangles in R_{s-1}. By construction, there are at least $T^3(n)-1$ points between two rectangles in R_q. Therefore, due to its height, the q-sized region must enclose at least $T^2(n)$ points in between each pair of rectangles it crosses in R_{s-1}. Thus the q-sized region contains at least $T^2(n)(T^2(n)-1) = T^4(n) - T^2(n)$ points. Therefore it cannot be used in a partition. This contradicts the assumption on its width, and thus, by definition, it cannot be s-sized. $\square$

Lemma 3 *Let Q be the number of classes of rectangles in the construction. For any solution to the range query problem using $O(T(n))$ time, the following must hold: for each $q \leq Q$, all but at most $1/T(n)$ of the universe must be covered by q-sized regions represented by the solution.*

Proof Assume that we chose to use an arbitrary rectangle in R_q as a query region. Then, in order to perform our query in time $T(n)$ our data structure must contain regions such that any rectangle in R_q can be covered by at most $T(n)$ regions.

By the construction of the example, all but a negligible part of the universe is covered by each set R_q. It thus suffices to show that for each rectangle in R_q, only $1/T(n)$ of the rectangle may not be covered by q-sized regions in the solution. From Lemma 2 and the definition of q-sized, it follows that any s-sized region, $(s \neq q)$, covers less than $1/T^3(n)$ of the query rectangle. Thus, $\Theta(T(n))$ non-q-sized regions cover $O(1/T^2(n))$ of the query rectangle. Therefore, the remainder of the query rectangle must be covered by q-sized regions. If not, the cost of a query would exceed $\Theta(T(n))$. $\square$

Theorem 1 *In order to search in $O(T(n)+k)$ time, where k is the number of elements in the query region, $\Omega\left(\frac{\log n}{\log T(n)}\right)$ partitions are required.*

Proof Let U be the area of the universe, Q be the number of classes of rectangles in the construction and A_q be the total area of q-shaped regions in all partitions. Since the area of one partition is U, then the total number of partitions is at least $\left(\sum_{q=1}^{Q} A_q\right)/U$. Lemma 3 implies that $A_q \geq (T(n)-1)/T(n)$ and the proof follows. $\square$

The theorem yields the following corollary.

Corollary 1 *If each partition requires $\Theta(n)$ space, any algorithm based on layered partitions and supporting range reporting in $O(\log^c n + k)$, $c = O(1)$, time requires $\Omega\left(\frac{n \log n}{\log\log n}\right)$ space.*

4 Conclusion

We feel that our new lower bound provides new insight on the difficulty of range searching. Our model of computation, layered partitions, captures the inherent properties of a large class of data structures. In this model, we have shown that it is not possible to perform range queries in $O(\log^c n + k)$ time and linear space, unless partitions can be stored in compressed form. However, it seems infeasible to be able to apply the bit-encoding technique of M-structures [7] to other than orthogonal rectilinear queries, since the compression heavily exploits the fact that the queries are rectangular. When compression is not feasible, any algorithm based on layered partitions and supporting range reporting in $O(\log^c n + k)$, $c = O(1)$, time requires $\Omega\left(\frac{n \log n}{\log \log n}\right)$ space. Thus we show that no superior upper bound can be achieved using traditional methods and data structures for range searching in the general case.

One might also imagine that a possible way to achieve a better tradeoff between space and time would be to combine some classical data structure, such as range trees, with some sophisticated search method, such as the one used in fusion trees [11]. The number of possibilities seems to be very large and we believe that many researchers have tried methods like this. In this article, we have shown that such an approach would not be fruitful.

Acknowledgments

We note our gratitude to the anonymous referee for many insightful comments. We would like to thank the referees of the Workshop of Algorithms and Data Structures, WADS'95, and Ola Petersson.

References

[1] A. Andersson. Sublogarithmic searching without multiplications. In *Proc. 36th IEEE Symposium on Foundations of Computer Science*, pages 655–663. ACM Press, 1995.

[2] A. Andersson. Faster deterministic sorting and searching in linear space. To appear in Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci., 1996.

[3] J. L. Bentley. Decomposable searching problems. *Inform. Process. Lett.*, 8:244–251, 1979.

[4] J. L. Bentley and H. A. Maurer. Efficient worst-case data structures for range searching. *Acta Inform.*, 13:155–168, 1980.

[5] B. Chazelle. Filtering search: a new approach to query-answering. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 122–132, 1983.

[6] B. Chazelle. Slimming down search structures: A functional approach to algorithm design. In *Proc. 26th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 165–174, 1985.

[7] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17:427–462, 1988.

[8] B. Chazelle. Lower bounds for orthogonal range searching, I: the reporting case. *J. ACM*, 37:200–212, 1990.

[9] B. Chazelle. Lower bounds for orthogonal range searching: II. the arithmetic model. *Journal of the ACM*, 37(3):439–463, July 1990.

[10] R. A. Finkel and J. L. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Inform.*, 4:1–9, 1974.

[11] M. L. Fredman and D. E. Willard. Blasting through the information theoretic barrier with fusion trees. In *Proc. 22nd Annu. ACM Sympos. Theory Comput.*, pages 1–7, 1990.

[12] G. S. Lueker. A data structure for orthogonal range queries. In *Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 28–34, 1978.

[13] E. M. McCreight. Priority search trees. *SIAM J. Comput.*, 14:257–276, 1985.

[14] P. B. Miltersen. Personal communication.

[15] P. B. Miltersen. Lower bounds for union-split-find related problems on random access machines. In *Proc. 26th Ann. ACM STOC*, pages 625–634, 1994.

[16] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, 1995. To appear.

[17] J. B. Saxe and J. L. Bentley. Transforming static data structures to dynamic structures. In *Proc. 20th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 148–168, 1979.

[18] D. E. Willard. A new time complexity for orthogonal range queries. In *Proc. 20th Allerton Conf. Commun. Control Comput.*, pages 462–471, 1982.