Lower Bounds for Orthogonal Range Searching: I. The Reporting Case

BERNARD CHAZELLE

Princeton University, Princeton, New Jersey

Abstract. We establish lower bounds on the complexity of orthogonal range reporting in the static case. Given a collection of n points in d-space and a box $[a_1, b_1] \times \cdots \times [a_d, b_d]$, report every point whose ith coordinate lies in $[a_i, b_i]$, for each $i = 1, \ldots, d$. The collection of points is fixed once and for all and can be preprocessed. The box, on the other hand, constitutes a query that must be answered online. It is shown that on a pointer machine a query time of $O(k + \text{polylog}(n))$, where k is the number of points to be reported, can only be achieved at the expense of $\Omega(n(\log n/\log\log n)^{d-1})$ storage. Interestingly, these bounds are optimal in the pointer machine model, but they can be improved (ever so slightly) on a random access machine. In a companion paper, we address the related problem of adding up weights assigned to the points in the query box.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*sorting and searching*; E.1 [**Data**]: Data Structures

General Terms: Algorithms, Theory

1. *Introduction*

Orthogonal range searching [1–5, 8–10, 12–14] is a direct generalization of list searching. Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in d-space; assume that each point is given by its coordinates in a fixed Cartesian system of reference. Given a query $q = [a_1, b_1] \times \cdots \times [a_d, b_d]$, report all the points in $P \cap q$. To motivate this problem, the classical scenario is to regard the points of P as the employees of a certain company. The coordinates of the points are important attributes of the employees, such as seniority, age, salary, traveling assignments, legal status, etc. Crucial to the well functioning of the company is the ability to provide fast answers to questions of the form: "*Which senior-level employees are less than x years of age, make more than y dollars a year, have been abroad in the last z months, have never taken the Fifth Amendment, etc.*?"

For obvious reasons, this is called orthogonal range searching in report-mode, or more simply, *orthogonal range reporting*. To put this problem in perspective, let

A preliminary version of this work has appeared in *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, New York, 1986, pp. 87–96.

Part of this research was done while B. Chazelle was a visiting professor at École Normale Supérieure, Paris, France. The research for this paper was partially supported by the National Science Foundation (NSF) under grant CCR 87-00917.

Author's address: Department of Computer Science, Princeton University, Princeton, N.J. 08544.

us define a *d-range* as the Cartesian product of d closed intervals over the real line. Orthogonal range reporting refers to the task of reporting all the points of $P \cap q$, given an arbitrary query d-range q. Evaluating the size of $P \cap q$ is called *orthogonal range counting*. Both problems can be put under the same umbrella by associating a weight $w(p)$ to each point $p \in P$. Weights are chosen in a suitable algebraic structure $(\mathscr{S}, +)$, such as a group or a semigroup. Given a query d-range q, *orthogonal range searching* is the problem of computing the sum $\sum_{p \in P \cap q} w(p)$. The group $(\mathscr{Z}, +)$ can be used for range counting, while the semigroup $(2^P, \cup)$ is suitable for the reporting version of the problem. The generality of range searching has other benefits. For example, given n points in 3-space, computing the highest point over a query rectangular region can be regarded as a two-dimensional range-searching problem.

From a complexity viewpoint, range reporting must be studied separately because, as opposed to, say, range counting, the query time is a function not only of the input size n, but also of the output size. The most efficient data structures to date capitalize on the fact that if many points are to be reported, the query time will be high, anyway, so the search component of the query-answering process does not need to be all that efficient; this is the idea of *filtering search* [3], which is behind many algorithms for multidimensional searching. To introduce our main result, let us consider the case $d = 2$. As was shown in [3], there exists a data structure of size $O(n \log n / \log \log n)$[1] that guarantees a query time of $O(k + \log n)$, where k is the number of points to be reported. The algorithm operates on a *pointer machine* [11], which means that no address calculations are required. The factor $\log n / \log \log n$ might look a little odd, especially in light of the fact that it can be removed in many variants of the problem. Indeed, there exist data structures of size $O(n)$ and query time $O(k + \log n)$ if (i) the query rectangle is *grounded*, meaning that it is required to be in contact with a fixed line [9], or (ii) its aspect-ratio is fixed [5], or (iii) the query is a grounded trapezoid [6]. Again, in all cases, those linear-size solutions work on a pointer machine. Adding to the suspicion that the factor $\log n / \log \log n$ can be removed (or at least lowered) on a pointer machine is the fact that it can on a random access machine (RAM). If address calculations are allowed, then the storage requirement can be reduced to $O(n \log^{\epsilon} n)$, for any fixed $\epsilon > 0$ [4].

Putting these questions to rest, we prove the rather surprising result that if a query time of $O(k + \log n)$ is desired then the storage must be $\Omega(n \log n / \log \log n)$. More generally, we show that in d dimensions, a query time of $O(k + \log^c n)$, for *any* constant c, can only be achieved at the expense of $\Omega(n(\log n / \log \log n)^{d-1})$ storage, which is optimal. This comes in sharp contrast with the random access machine model, in which a query time of $O(k + \log^{d-1} n)$ can be achieved with only $O(n(\log n)^{d-2+\epsilon})$ storage [4]. These results show that orthogonal range reporting is inherently more difficult on a pointer machine than on a RAM. This is a rare separation result that sheds light on two of the most common models of sequential computation.

The lower-bound proof is based entirely on control-flow considerations and relies on syntactic, rather than semantic, properties of pointer machine-based data structures. We believe that the underlying idea can be used for many other problems. As a starter, we recall a few basic facts about pointer machines and prove a result of general interest concerning the complexity of navigating through a

[1] All logarithms in this paper are taken to the base 2, unless specified otherwise.

pointer-based data structure (Section 2). In two dimensions, the desired lower bound can be established by constructive means (Section 3). Things become a little more complicated in higher dimensions, so we turn to probabilistic arguments to prove the existence of "hard" inputs (Section 4). The issue of matching upper bounds is addressed in Section 5. We close with concluding remarks in Section 6.

2. *The Complexity of Navigation on a Pointer Machine*

We assume that the reader is familiar with the notion of a pointer machine, as defined in [11]. As long as we are only concerned with lower bounds, we have the freedom to increase the power of our machine at will. This is a good idea aimed at eliminating unessential particulars that might get in the way of a simple proof. The memory of a pointer machine is an unbounded collection of registers. Each register is a record with a fixed number of data and pointer fields. In this way, the memory can be modeled as a directed graph with bounded outdegree. At the cost of increasing the storage requirement by a constant multiplicative factor, we can always reduce the outdegree of each node of the graph to 0, 1, or 2 (why?).

Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in E^d. A data structure for orthogonal range reporting is a digraph $G = (V, E)$ with a source σ. Each node v is assigned an integer, $label(v)$, between 0 and n. If $i = label(v)$ is not 0, the node v is associated with point p_i. Note that many nodes can be labeled the same way. Given a query q, the answering algorithm begins a traversal of G at the source σ. We place only two restrictions on the algorithm. To begin with, no node (other than the source) can be visited if a node pointing to it has not already been visited. Furthermore, completion cannot occur until each label i ($p_i \in q$) has been encountered at least once. During its execution, the algorithm is allowed to modify data and address fields as it visits them. It can also add new nodes to G by requesting them from a pool of free nodes with blank fields, the *freelist*. Let $N(v)$ denote the set $\{w \mid (v, w) \in E\}$. For our purposes, the algorithm executes a program made up of the following instructions. Initially, $W = \{\sigma\}$:

(i) Pick any $v \in W$ and add $N(v)$ to W.
(ii) Request a new node v from the freelist and add it to W. Initialize $N(v)$ to $\emptyset$.
(iii) Pick any $v, w \in W$ and, if $|N(v)| < 2$, add (v, w) to E (and v or w to V, if necessary).
(iv) Pick any $v, w \in W$ and remove the edge (v, w) from E if it exists.

Let $W(q)$ denote the set W at termination. Correctness is ensured by the requirement

$$\{i \mid p_i \in q\} \subseteq \{label(v) \mid v \in W\}.$$

This means that the query-answering algorithm must reach at least one representative node for every point to be reported. The query time is measured as $|W(q)|$.

Let us make a few comments about this framework. Our first remark is that the model is more powerful than the standard pointer machine. (Once again, recall that this is a *good* thing in the context of lower bounds.) The "*pick any*" feature cannot be implemented in constant time on a real computer. After a while, W can grow to be very big, and "*pick any*" offers the full power of nondeterminism. The

fact that we add all of $N(v)$ into W, as opposed to any particular node is inconsequential, considering that $|N(v)| \leq 2$. The labeling scheme has a natural interpretation. Nodes labeled 0 are used internally by the data structure. The other nodes are used both internally but also as means of output. Suppose that $p_i \in q$. Then, the model requires the algorithm to reach at least one node labeled i. This assumption might seem unnecessarily exclusive: After all, why can't we simply require that i be computed rather than discovered? Certainly, the output should give us direct access to the points of $P \cap q$. However, the mere knowledge of their indices does not fulfill this requirement on a pointer machine. This is perhaps where the difference between a RAM and a pointer machine shines in its true light. If p_i lies in q, what we want is either p_i itself or a pointer to p_i, but not just the integer i. In the model described above, we can store p_i inside each record labeled i. If the coordinates of p_i are huge or are defined implicitly, then we might want to keep P in a table and add a pointer from each record labeled i to the entry of the table where p_i is stored. This model describes all the pointer machine data structures proposed in the literature for solving range-searching problems.

Let a and b be two positive reals. We say that the data structure $G = (V, E)$ for P is (a, b)-*effective* if, for any query d-range q, we have $|W(q)| \leq a(|P \cap q| + \log^b n)$. This means that the query time is linear in the output size, aside from a polylogarithmic overhead. To establish our lower bound, intuitively, we want to argue that nodes labeled after the points of $P \cap q$ cannot be too much spread out in the graph; otherwise, it would be impossible to get to them in linear time. Our goal is to produce a large set of queries, no two of which share too many points of P. In this way, we are able to argue that the graph has many subsets of nodes with the following properties: (i) each subset is fairly compact (vertices are close to each other), and (ii) no two subsets share too many edges. This will imply that the graph has many edges. Now, by virtue of the bounded outdegree condition, this means that the graph has many vertices, and therefore the need for storage is large. Let $S = \{q_1, \ldots, q_s\}$ be a set of queries and let α be a positive real. We say that S is α-*favorable* if, for each i, j $(i \neq j)$,

(i) $|P \cap q_i| \geq \log^\alpha n$.
(ii) $|P \cap q_i \cap q_j| \leq 1$.

The following result is the key to the lower-bound proof. It asserts that the storage requirement is at least proportional to the size of any b-favorable set of queries. The lemma suggests the remainder of the proof: finding large favorable sets of queries. Interestingly, the investigation of favorable sets is purely geometric, and does not involve the model of computation at all.

LEMMA 2.1. *Let a and b be two positive reals. If the data structure G is (a, b)-effective and the set of queries S is b-favorable, then $|V| > |S|(\log^b n)/2^{16a+4}$, for n large enough.*

PROOF. We define the distance $p(v, w)$ between two nodes v and w of G as the number of edges on the shortest directed path from v to w. If there is no such path, $p(v, w) = +\infty$. The distance $p(v, w)$ has serious deficiencies: It is not symmetric and it is not always finite. We correct for these flaws by introducing the new distance $d(v, w) = \min\{p(z, v) + p(z, w) \mid z \in V\}$. Note that because of the source σ the new distance is always finite. Given a query q we define *orbit*(q) as the set

of pairs $(v, w) \in V^2$ such that

(i) $label(v) \in \{i \mid p_i \in q\}$,
(ii) $label(w) \in \{i \mid p_i \in q\}$,
(iii) $label(v) < label(w)$,
(iv) $d(v, w) < 8a$.

Note that the pair (v, w) need not be an edge of G. The various conditions express the fact that (i)–(ii) both v and w can be used for the output, (iii) the pair is neither trivial nor repeated, (iv) v and w are within a constant distance of each other.

Given a query q, consider the graph G *before* the algorithm begins. There exists at least one path from the source σ to every node of $W(q)$. This implies the existence of a directed tree T rooted at the source, whose node-set is $W(q)$ (the reader will forgive us for leaving these facts as exercises). The tree T is called directed because all its edges are oriented in the same direction (pointing away from the root). Let $v_1, v_2, \ldots, v_m$ be the sequence of nodes of the tree T corresponding to a first-order traversal. For each index in $\{i \mid p_i \in q\}$, mark exactly one node v_j such that $label(v_j) = i$. Let $w_1, w_2, \ldots, w_\ell$ be the subsequence of marked v_i's. Note that $m = |W(q)|$ and $\ell = |P \cap q|$. Next, we define a distance $\delta(v, w)$ over the tree T in exactly the same way we defined $d(v, w)$ over G. If v and w belong to T, it is clear that

$$d(v, w) \leq \delta(v, w) < +\infty. \tag{1}$$

The distance δ trivially satisfies the triangular inequality. Therefore

$$\sum_{1 \leq i < \ell} \delta(w_i, w_{i+1}) \leq \sum_{1 \leq i < m} \delta(v_i, v_{i+1}). \tag{2}$$

The distance $\delta(v_i, v_{i+1})$ measures the number of edges on the path(s) from z to v_i and v_{i+1}, where z is the nearest common ancestor of v_i and v_{i+1} in T. Because of the first-order labeling, the sum $\sum_{1 \leq i < m} \delta(v_i, v_{i+1})$ counts each edge of T at most twice. Therefore,

$$\sum_{1 \leq i < m} \delta(v_i, v_{i+1}) \leq 2(|W(q)| - 1). \tag{3}$$

Since G is (a, b)-effective, we have $|W(q)| \leq a(|P \cap q| + \log^b n)$. Assume now that $|P \cap q| \geq \log^b n$. We derive

$$|W(q)| \leq 2a|P \cap q|. \tag{4}$$

Combining (1)–(4), we find

$$\sum_{1 \leq i < \ell} d(w_i, w_{i+1}) < 4a|P \cap q|,$$

and hence, $|\{i \mid d(w_i, w_{i+1}) \geq 8a\}| < |P \cap q|/2$. Since $\ell = |P \cap q|$, we find that

$$|\{i \mid d(w_i, w_{i+1}) < 8a\}| > \frac{|P \cap q|}{2} - 1. \tag{5}$$

Let $S = \{q_1, \ldots, q_s\}$ be a b-favorable set of queries. Relation (5) holds for each q_i. Moreover, because no two points of P can belong to two distinct queries, the pairs (w_i, w_{i+1}) are all distinct. Since $|P \cap q|$ goes to infinity with n,

$$\left| \bigcup_{1 \leq i \leq s} orbit(q_i) \right| = \sum_{1 \leq i \leq s} |orbit(q_i)| > \frac{1}{4} |S| \log^b n, \tag{6}$$

for n large enough. Because of the bounded outdegree condition ($|N(v)| \leq 2$, for each $v \in V$), we have

$$|\{(v, w) \in V^2 \mid d(v, w) < 8a\}| \leq |\{(z, v, w) \in V^3 \mid p(z, v) < 8a \text{ and } p(z, w) < 8a\}| \leq |V| 2^{16a+2}.$$

From (6) it follows that $|V| > |S|(\log^b n)/2^{16a+4}$, which completes the proof. □

Lemma 2.1 suggests an obvious line of attack for proving lower bounds on the size of the data structure. Since V is already a lower bound, it suffices to exhibit a large b-favorable set of queries. In the two-dimensional case (Section 3), we are able to define the points and queries by constructive means. In higher dimensions (Section 4), we can still define the queries explicitly, but we must rely on probabilistic arguments to produce a *hard* set of points.

3. *Orthogonal Range Reporting in the Planar Case*

Although this section is superseded by the next one, we include it because it gives us a glimpse of what "hard" inputs look like. We define the points first; then we specify the queries and show that they form a b-favorable set. Let $m = \lfloor 2 \log^b p \rfloor$ and $\lambda = \lfloor (\log p)/(1 + b \log\log p) \rfloor$, where p is an integer large enough so that m, $\lambda > 4$. We define n as m^λ.[2] Each query will contain exactly m points. The definition of the queries will be recursive, which explains why the choice of n as a power of m is particularly appropriate. For p large enough, it is easy to show that $m \geq \log^b n$ and

$$\lambda \geq \left\lfloor \frac{\log n}{1 + b \log\log n} \right\rfloor.$$

For any integer i ($0 \leq i < n$) let $\rho_m(i)$ be the integer obtained by writing i in base m over λ bits and reversing the digits. If $i = m_1 m_2 \cdots m_\lambda$, we have $\rho_m(i) = m_\lambda \cdots m_2 m_1$. We now define the set of points P as $\{(\rho_m(i), i) \mid 0 \leq i < n\}$. This definition generalizes the bit-reversal permutation. Figure 1 illustrates the case $n = 3^3$. Keep in mind that the point-set is two-dimensional: Edges have been drawn to exhibit the structure of the point system.

We are now ready to define the queries. Going back to Figure 1, we notice that the cube has three principal 3-point sequences: (a, b, c), (a, d, e), and (a, f, g). Each sequence can be enclosed tightly by a query rectangle. Since we have 9 translates of each sequence, we obtain a total of 27 queries. In general, we have $\lambda m^{\lambda-1}$ queries. To see this, think of a tree T that encodes the x-coordinates of P in base m. Each internal node of T has m children, labeled $0, 1, \ldots, m-1$ from left to right. The tree has depth λ. The root of T is associated with P as a whole. A node v with ancestors labeled $m_1, m_2, \ldots, m_r$, from the root down, maps to the subset of points whose x-coordinates have the prefix $m_1 m_2 \cdots m_r$ in base m. For each internal node v, sort its associated set of points by y-coordinates and break it up into groups of size m: Each group can be enclosed by a query rectangle that does not contain any other point. Since a node with r ancestors has n/m^r associated

[2] Restricting n to be a power of m should not be viewed as a weakness of the argument. For one thing, the proof can be generalized to any n. Also, if we are concerned with lower bounds, we should be (reasonably) happy as long as we have an infinite number of input sizes for which the lower bound holds. But there is a third, more compelling reason: the results of the next section will supersede the results of this one, so why bother?

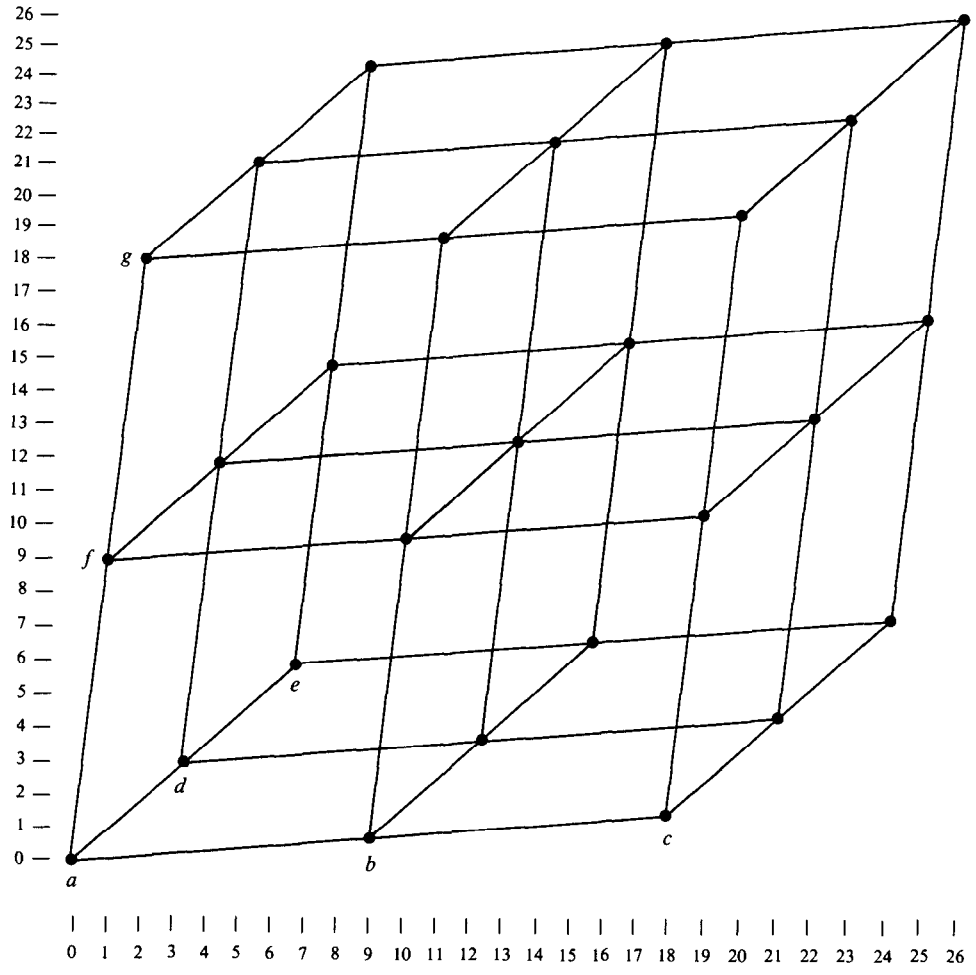

FIGURE 1

points, this will give us a total of

$$\sum_{0 \le r < \lambda} (\#\text{ nodes at depth } r) \times \frac{n/m^r}{m} = \frac{\lambda n}{m}$$

queries. It is immediate to check that, for n large enough, the set of queries is b-favorable. From Lemma 2.1 and the fact that $m = O(\log^b n)$, we conclude that a query time of $O(|P \cap q| + \text{polylog}(n))$ can be achieved only at a cost of $O(n \log n / \log\log n)$ storage. It follows from [3] that this lower bound is tight on a pointer machine.

THEOREM 3.1. *Given n points in E^2 there exists a data structure of size $O(n \log n / \log\log n)$ that allows us to answer any orthogonal range reporting query in time $O(k + \log n)$, where k is the number of points to be reported. The algorithm is optimal on a pointer machine.*

4. *Orthogonal Range Reporting in Higher Dimensions*

We now investigate the complexity of the problem in E^d ($d > 1$). We use the notation λ_d to denote the Lebesgue measure in E^d. To begin with, we construct a set of candidate queries by interpreting the discrete criteria of b-favorability in "continuous" terms. In a second stage, we show that throwing in points at random makes a large subset of our queries b-favorable with high probability. To simplify the notation, we introduce the following parameters:

$$\begin{cases} \alpha = \left(\dfrac{4\log^b n}{n}\right)^{1/d} \\ \mu = \dfrac{1}{(\log n)^{b+d}} \\ \delta = \dfrac{4}{n\log^d n}. \end{cases}$$

The queries defined in Section 3 for the two-dimensional case can be regarded as the cells of λ lattices; each lattice corresponds to a distinct level in the underlying tree T. The nice total order among lattices will be lost in dimension $d > 2$, so we will use a multi-index $z \in \mathcal{Z}^d$ to refer to any particular lattice. The index z is a point on the discrete hyperplane

$$\mathcal{T} = \left\{(z_1, \ldots, z_d) \in \mathcal{Z}^d \,\middle|\, \sum_{1 \le k \le d} z_k = 0\right\}.$$

Within a given lattice, a particular cell will be referred to by the integer coordinates of one of its corners, expressed as a vector $j \in \mathcal{N}^d$. For each $z = (z_1, \ldots, z_d) \in \mathcal{T}$ and $j = (j_1, \ldots, j_d) \in \mathcal{N}^d$, let $v(z; j)$ denote the d-range

$$\prod_{1 \le k \le d} [j_k \alpha \mu^{z_k}, (j_k + 1)\alpha\mu^{z_k}].$$

Our set of candidate queries is defined as

$$\mathcal{G} = \{v(z; j) \subseteq [0, 1]^d \mid (z, j) \in \mathcal{T} \times \mathcal{N}^d\}.$$

The elements of $\mathcal{G}$ are called *cells*. They all have the same (Lebesgue) measure, equal to α^d. We need three technical results. Lemma 4.1 shows that two cells cannot overlap too much. This is the continuous equivalent of condition (ii) of favorable sets, that is, $|P \cap q_i \cap q_j| \le 1$. Lemma 4.2 asserts that the set of candidate queries is not too small. Finally, Lemma 4.3 states that only a few cells can intersect in one point. This will be useful for a later mop-up operation. The idea is that some of the points thrown at random will be judged *bad* and thus ignored. The lemma says that a single bad point cannot affect too many candidate queries.

LEMMA 4.1. *For any n large enough, the intersection of two distinct cells has measure at most δ.*

PROOF. Let $c = v(z; j)$ and $c' = v(z'; j')$ be two distinct cells. If $z = z'$, then $j \neq j'$ and obviously $\lambda_d(c \cap c') = 0$. If now $z \neq z'$, we can assume without loss of generality that $z_1 \neq z_1'$, with, say, $z_1 < z_1'$. Let c_k (resp., c_k') be the projection of c (resp., c') on the x_k-axis ($1 \le k \le d$). For n large enough, we have the

following derivations

$$\begin{aligned}\lambda_d(c \cap c') &= \prod_{1 \le k \le d} \lambda_1(c_k \cap c_k') \\ &\le \prod_{1 \le k \le d} \min\{\lambda_1(c_k), \lambda_1(c_k')\} \\ &\le \mu^{z_1' - z_1} \lambda_1(c_1) \times \prod_{1 < k \le d} \min\{\lambda_1(c_k), \lambda_1(c_k')\} \\ &\le \mu \lambda_d(c);\end{aligned}$$

therefore, $\lambda_d(c \cap c') \le \mu\alpha^d$. □

LEMMA 4.2. *For any n large enough, we have*

$$|\mathscr{G}| > \frac{n(\log n)^{d-b-1}}{(b+d)^{4d}(\log\log n)^{d-1}}.$$

PROOF. We easily check that $\alpha\mu^{z_k} \le \frac{1}{2}$, for any $z_k \in \mathscr{Z}$ such that

$$z_k \ge \frac{1}{b+d}\left(\frac{1+2/d}{\log\log n} + \frac{b}{d} - \frac{\log n}{d\log\log n}\right). \tag{7}$$

This shows that, for any $z = (z_1, \ldots, z_d) \in \mathscr{T}$ such that each z_k satisfies (7), the number of cells of the form $v(z; j)$ is at least

$$\prod_{1 \le k \le d} \left\lfloor \frac{1}{\alpha\mu^{z_k}} \right\rfloor > \prod_{1 \le k \le d} \left(\frac{1}{\alpha\mu^{z_k}} - 1\right) \ge \frac{1}{(2\alpha)^d} = \frac{n}{2^{d+2}\log^b n}.$$

The right-hand side of (7) is negative for n large enough. This gives us the following lower bound on the number of $z \in \mathscr{T}$ whose coordinates z_k's satisfy (7):

$$\left(\left\lfloor \frac{1}{b+d}\left(\frac{\log n}{d\log\log n} - \frac{1+2/d}{\log\log n} - \frac{b}{d}\right)\right\rfloor + 1\right)^{d-1} > \left(\frac{\log n}{2d(b+d)\log\log n}\right)^{d-1}.$$

This implies that the number of cells in $\mathscr{G}$ exceeds

$$\frac{n(\log n)^{d-b-1}}{2^{2d+1}d^{d-1}(b+d)^{d-1}(\log\log n)^{d-1}},$$

from which the lemma follows. □

LEMMA 4.3. *For any n large enough, no point can lie in the interior of more than* $(d\log n/\log\log n)^{d-1}$ *cells of* $\mathscr{G}$.

PROOF. Suppose that $v(z; j) \subseteq [0, 1]^d$, for some $z = (z_1, \ldots, z_d) \in \mathscr{Z}^d$ and $j \in \mathscr{N}^d$. Then, for n large enough, z_k exceeds $-\frac{1}{2}\log n/\log\log n$ $(1 \le k \le d)$. The lemma follows from the fact that since $z \in \mathscr{T}$ we have $z_k < (d-1)/2 \log n/\log\log n$. □

The first criterion of b-favorability concerns the fact that a query should contain at least $\log^b n$ points of P. We strengthen this requirement a little to shield us against the later removal of *bad* points. We say that a cell is *heavy* if its interior contains more than $2\log^b n$ points of P. As usual, P denotes the set of n input points. A *random* set of points in a compact set K is a shorthand for a set of points whose elements have been chosen randomly and independently from the uniform distribution in K.

LEMMA 4.4. *Let P be a random set of n points in* $[0, 1]^d$. *If n is large enough, then with probability greater than* $1 - 2/\log^b n$, *more than half the cells of* $\mathscr{G}$ *are heavy.*

PROOF. Let c be an arbitrary cell of $\mathcal{G}$ and let χ be the number of points of P in the interior of c. We also define π as the probability that c is heavy (i.e., $\Pr(\chi > 2\log^b n)$). The mean and variance of χ are, respectively, $n\lambda_d(c)$ and $n\lambda_d(c)(1 - \lambda_d(c))$. Using Chebyshev's inequality [7], we derive

$$1 - \pi \leq \frac{n\lambda_d(c)(1 - \lambda_d(c))}{(n\lambda_d(c) - 2\log^b n)^2},$$

and hence $\pi > 1 - 1/\log^b n$. Let Π be the probability that more than half the cells of $\mathcal{G}$ are heavy. Since the expected number of heavy cells is equal to $\pi|\mathcal{G}|$, a straightforward Markov-type argument gives us

$$\left(1 - \frac{1}{\log^b n}\right)|\mathcal{G}| < \pi|\mathcal{G}| \leq \frac{1}{2}(1 - \Pi)|\mathcal{G}| + \Pi|\mathcal{G}|,$$

which completes the proof. □

To satisfy the second favorability criterion, we make sure that no two points are too close to each other. The underlying metric is not the Euclidean metric. We define the *range-distance* between two points $(x_1, \ldots, x_d)$ and $(y_1, \ldots, y_d)$ in E^d to be the measure of the smallest d-range containing them, that is, $\prod_{1 \leq k \leq d} |x_k - y_k|$. We say that a point $p \in P$ is *stranded* if its range-distance to any point of $P \setminus \{p\}$ exceeds δ. Finally, we say that P is *ρ-diffuse* ($0 \leq \rho \leq 1$) if it contains at least ρn stranded points.

LEMMA 4.5. *Let $\rho = 1 - 1/\sqrt{\log n}$. For any n large enough, a random set of n points in $[0, 1]^d$ is ρ-diffuse with probability greater than $1 - 2^{d+3}/\sqrt{\log n}$.*

PROOF. Let $L(d, y)$ denote the Lebesgue measure of $\{(x_1, \ldots, x_d) \in [0, 1]^d \mid \prod_{1 \leq k \leq d} x_k \leq y\}$, with $0 < y < 1$. We have the recurrence relation $L(0, y) = 0$ and, for $d \geq 1$,

$$L(d, y) = y + \int_y^1 L\left(d - 1, \frac{y}{x}\right) dx.$$

Let $M(d, y) = L(d, y) - L(d - 1, y)$, for $d \geq 1$. We derive the simpler recurrence: $M(1, y) = y$ and

$$M(d, y) = \int_y^1 M\left(d - 1, \frac{y}{x}\right) dx,$$

for $d > 1$. This gives

$$M(d, y) = y \int_1^{1/y} M\left(d - 1, \frac{1}{x}\right) dx,$$

hence,

$$M(d, y) = \int_1^{1/y} \int_1^{x_1} \cdots \int_1^{x_{d-2}} \frac{y}{x_1 \times \cdots \times x_{d-1}} dx_{d-1} \cdots dx_1.$$

It is elementary to evaluate this integral directly. This leads to

$$L(d, y) = \sum_{0 \leq k < d} \frac{y}{k!} \left(\ln \frac{1}{y}\right)^k,$$

assuming that $0! = 1$. Since $1/\delta$ goes to infinity with n, we have, for n large enough,

$$L(d, \delta) < \frac{(-1)^{d-1} d\delta}{(d-1)!} (\ln \delta)^{d-1} < 2\delta (\log n)^{d-1};$$

hence,

$$L(d, \delta) < \frac{8}{n \log n}. \tag{8}$$

Consider the locus of points in $[0, 1]^d$ whose range-distance to a given point of $[0, 1]^d$ is at most δ. Let V be the largest measure of such a set. It is immediate that

$$V \leq 2^d L(d, \delta). \tag{9}$$

Let ν be the expected number of stranded points in P and let π be the probability that P is ρ-diffuse. Obviously,

$$\nu \leq (1 - \pi)\rho n + \pi n. \tag{10}$$

The probability that a given point of P is stranded exceeds $1 - nV$; therefore, $\nu > n(1 - nV)$. Combining this inequality with (8)–(10) completes the proof. □

Lemmas 4.4 and 4.5 imply that with probability $1 - o(1)$ a random pick P of n points is ρ-diffuse and makes more than half the cells of $\mathcal{G}$ heavy. Move outside of $[0, 1]^d$ every point of P that is not stranded, and let Γ be the cells of $\mathcal{G}$ whose interiors contain at least $\log^b n$ points after the moving.

LEMMA 4.6. *For n large enough, we have*

$$|\Gamma| > \frac{n(\log n)^{d-b-1}}{(b+d)^{5d} (\log \log n)^{d-1}}.$$

PROOF. Mark each heavy cell of $\mathcal{G}$ that, prior to the moving, has at least half of its interior points stranded. Since P is ρ-diffuse, Lemma 4.3 shows that the number of heavy cells left unmarked cannot exceed $d^{d-1} n (\log n)^{d-b-3/2} / (\log \log n)^{d-1}$, for n large enough. Because more than half the cells are heavy, the proof follows from Lemma 4.2. □

The cells of Γ satisfy the first criterion of b-favorability. Lemma 4.1 justifies the notion of a stranded point. No two cells of Γ can contain the same pair of points of P (after the moving). We derive that Γ forms a b-favorable set of queries. The lower bound is now within reach. From Lemmas 2.1 and 4.6, we conclude that a query time of $O(\text{output size} + \text{polylog}(n))$ can only be achieved at the expense of $\Omega(n(\log n/\log \log n)^{d-1})$ storage. Although the set of input points is not random (because of the moving), it is not difficult to see how to modify the proof so that the lower bound holds for a random set P. This means that the lower bound is not confined to some pathological configuration of points, but applies to almost any point system. Another remark concerns the fact that orthogonal range reporting is defined over the reals (points of E^d). It is easy to see, however, that the proof technique discriminates among point sets only by means of coordinate comparisons. This easily implies that hard inputs can be defined with integer coordinates over $O(\log n)$ bits.

5. *Upper Bounds*

Assume that our pointer machine has only $O(n(\log n/\log \log n)^{d-1})$ storage. How fast can we solve orthogonal range reporting in d-space? We already know

the answer in the case where $d = 2$ (Theorem 3.1). In [10, pp. 47], Mehlhorn gives a general method for answering orthogonal range queries in time $O(k + (2^m/m)^{d-1}\log^d n)$, where k is the number of points to be reported, using a data structure of size $O(n(\log n)^{d-1}/m^{d-1})$. The quantity m is a *slack* parameter that we can adjust at will. The assignment $m = \lceil \epsilon \log\log n/(d-1)\rceil$, for any fixed $\epsilon > 0$, guarantees that the space requirement stays within the desired bounds. The query time is $O(k + \log^{d+\epsilon} n)$.

As it turns out, Mehlhorn's data structure was designed to support dynamic operations, and it can be slimmed down a little if it is to be used in a static environment. For the sake of completeness, we briefly sketch how this works. We assume that the reader is familiar with the notion of a *range tree* [1]. We briefly recall the definition. Let $q_1, \ldots, q_n$ be a set of n points in the plane. We can always relabel the points so that $q_1, \ldots, q_n$ appear in order of nondecreasing x-coordinates (breaking ties arbitrarily). Let T be a binary tree whose leaves are associated bijectively with $q_1, \ldots, q_n$, from left to right. For each node v of the tree we define the *node-set* $N(v)$ as the set of points associated with the leaves descending from v. The root has $\{q_1, \ldots, q_n\}$ as node-set, while each leaf has a singleton. There is more to the story of range trees, but this is enough for our purposes.

Returning now to orthogonal range reporting, we build a range tree with respect to the set $\{p_1, p_2, \ldots, p_n\}$ along the first dimension. For each node v of the tree, let $N^*(v)$ denote the set of $(d-1)$-dimensional points obtained by depriving each point of $N(v)$ of its first coordinate. Next, for each node v of the tree whose depth is a multiple of the slack parameter m, attach a pointer to a data structure defined recursively with respect to $N^*(v)$. Note that the dimension of the sets decreases by 1 with each recursive call. When the dimension is 2, instead of pushing the recursion further, we simply attach a pointer to the two-dimensional data structure of Theorem 3.1.

From $d = 2$ on, each increment of 1 in dimension increases the search component of the query time by a factor of $O(2^m(\log n)/m)$. Similarly, the storage is multiplied by a factor of $O((\log n)/m)$. The same assignment of the slack parameter gives us a data structure of the desired size, $O(n(\log n/\log\log n)^{d-1})$. Its query time is in $O(k + \log^{d-1+\epsilon} n)$.

THEOREM 5.1. *Consider a data structure for orthogonal range reporting on n points in E^d that operates on a pointer machine, and let c be an arbitrary constant. If the data structure provides a query time of $O(k + \log^c n)$, where k is the number of points to be reported, then its size must be $\Omega(n(\log n/\log\log n)^{d-1})$. The lower bound is tight for any $c \geq 1$, if $d = 2$, and any $c \geq d - 1 + \epsilon$ (for any $\epsilon > 0$), if $d > 2$.*

6. *Conclusions and Open Problems*

Theorem 5.1 says that, for the query time claimed, no data structure can be asymptotically smaller than stated. It does not say, however, that the query time is optimal. To reduce the upper bound in dimension greater than 2 is an interesting open problem. Another limitation of our proof technique is to require the query time to be linear in the output size. If instead of $O(k + \log^c n)$, we allow a more general function such as $O(f(k, n) + \log^c n)$, what can we say? If $f(n, k)$ is small, say, $O(k\log\log n)$, the same technique will lead to a nontrivial lower bound. If $f(k, n) = k\log n$, however, the technique breaks down completely. It is easy to see why. Store the points in the nodes of a perfectly balanced binary tree. Every point of the output is at most $\log n$ nodes away from the root; therefore, $\Omega(n)$ is the only

lower bound on the storage requirement we can expect to derive. What goes wrong here is that our model of computation charges only for the number of memory cells visited, and not for the time spent figuring out which cells to visit. This is good in the sense that lower bounds have very broad applicability. On the other hand, more cost factors may sometimes be needed in order to obtain nontrivial lower bounds. This is certainly the case if we are interested in query times of the form $O(k \log n + \log^c n)$.

Editor's Note: Part II of Dr. Chazelle's paper will appear in the July issue of *J. ACM*.

ACKNOWLEDGMENTS. I wish to thank Robert E. Tarjan for valuable comments on a draft of this paper. I am also grateful to the referees for several useful suggestions.

REFERENCES

1. BENTLEY, J. L. Decomposable searching problems. *Inf. Proc. Lett. 8* (1979), 244–251.
2. BENTLEY, J. L. Multidimensional divide and conquer. *Commun. ACM 23*, 4 (1980), 214–229.
3. CHAZELLE, B. Filtering search: A new approach to query-answering. *SIAM J. Comput. 15*, 3 (1986), 703–724.
4. CHAZELLE, B. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput. 17*, 3 (1988), 427–462.
5. CHAZELLE, B., AND EDELSBRUNNER, H. Linear space data structures for two types of range search. *Discrete Comput. Geom. 2* (1987), 113–126.
6. CHAZELLE, B., AND GUIBAS, L. J. Fractional cascading: II. Applications. *Algorithmica 1* (1986), 163–191.
7. FELLER, W. *An Introduction to Probability Theory and Its Applications*, vol. 1, 3rd ed. Wiley, New York, 1968.
8. LUEKER, G. S. A data structure for orthogonal range queries. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, New York, 1978, pp. 28–34.
9. MCCREIGHT, E. M. Priority search trees. *SIAM J. Comput. 14*, 2 (1985), 257–276.
10. MEHLHORN, K. *Data Structures and Algorithms. 3: Multidimensional Searching and Computational Geometry*. Springer-Verlag, Berlin, 1984.
11. TARJAN, R. E. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. Comput. System Sci. 18* (1979), 110–127.
12. WILLARD, D. E. Predicate-oriented database search algorithms. Rep. TR-20-78. PhD dissertation, Aiken Computational Lab., Harvard Univ., Cambridge, Mass., 1978.
13. WILLARD, D. E. New data structures for orthogonal range queries. *SIAM J. Comput. 14*, 1 (1985), 232–253.
14. WILLARD, D. E., AND LUEKER, G. S. Adding range restriction capability to dynamic data structures. *J. ACM 32*, 3 (1985), 597–617.

RECEIVED AUGUST 1986; REVISED JUNE 1988; ACCEPTED JUNE 1989