Improved Space Bounds for Cache-Oblivious Range Reporting

Peyman Afshani* Norbert Zeh†

Abstract

We provide improved bounds on the size of cache-oblivious range reporting data structures that achieve the optimal query bound of $O(\log_B N + K/B)$ block transfers. Our first main result is an $O(N\sqrt{\log N} \log\log N)$-space data structure that achieves this query bound for 3-d dominance reporting and 2-d three-sided range reporting. No cache-oblivious $o(N \log N/ \log\log N)$-space data structure for these problems was known before, even when allowing a query bound of $O(\log_2^{O(1)} N + K/B)$ block transfers.[1] Our result also implies improved space bounds for general 2-d and 3-d orthogonal range reporting. Our second main result shows that any cache-oblivious 2-d three-sided range reporting data structure with the optimal query bound has to use $\Omega(N \log^{\varepsilon} N)$ space, thereby improving on a recent lower bound for the same problem. Using known transformations, the lower bound extends to 3-d dominance reporting and 3-d halfspace range reporting.

1 Introduction

Range searching is one of the most fundamental problems in computational geometry. Given a set S of N points in $\mathbb{R}^d$, the task is to preprocess S so that all points in a query region can be counted (*range counting*) or reported (*range reporting*) efficiently. *Approximate range counting* asks for an approximation of the number, K, of points in the query range that is no less than K and no greater than $(1 + \varepsilon)K$, for some $\varepsilon > 0$. Typical range searching problems are expressed in more specific terms depending on the shape of the query: simplices, halfspaces, circles, and axis-aligned boxes give rise to *simplex range searching*, *halfspace range searching*, *circular range searching*, and *orthogonal range searching* problems, respectively; see Figure 1.

Most previous work on range searching focused on *internal memory* models of computation, such as the RAM model or the pointer machine model. While these models are useful for studying the fundamental computational complexity of a problem, they ignore that modern computers are equipped with memory hierarchies whose access times vary by factors of up to 10^6 depending on the memory level currently holding the accessed data item. Among the models proposed to capture these varying access costs in real memory hierarchies, the *input-output model* (or *I/O model*) [7] and the *cache-oblivious model* [14] are the most widely accepted ones.

In the I/O model, the computer is equipped with two levels of memory: a slow but conceptually unlimited *external memory* and a fast *internal memory* with capacity M. All computation happens on data in internal memory. Data is transferred between internal and external memory in blocks of B consecutive data items. The complexity of an algorithm is the number of such *block transfers* it performs.

The cache-oblivious model provides a simple framework for designing algorithms for *multi-level* memory hierarchies. In this model, the algorithm is oblivious of the details of the memory hierarchy but is analyzed in the I/O model, assuming the block transfers necessary to bring the data accessed by the algorithm into memory are performed by an *offline optimal* paging algorithm, that is, one that performs the minimum number of block transfers for the data access sequence of the algorithm. Since the algorithm is designed without reference to M or B, the analysis can be applied to *any* two levels of a multi-level memory hierarchy. In particular, if the analysis shows that the algorithm incurs an optimal number of block transfers with respect to two levels of the memory hierarchy, it does so simultaneously at all levels. See [14] for a more detailed discussion of the model.

In this paper, we focus mostly on cache-oblivious

*Faculty of Computer Science, Dalhousie University, Halifax, NS B3H 1W5, Canada. Email: peyman@madalgo.au.dk. This research was done while the first author was a postdoctoral fellow at the MADALGO Center for Massive Data Algorithmics, Department of Computer Science, Aarhus University, Denmark and was supported in part by the Danish National Research Foundation and the Danish Strategic Research Council.

†Faculty of Computer Science, Dalhousie University, Halifax, NS B3H 1W5, Canada. Email: nzeh@cs.dal.ca. This research was supported in part by NSERC and the Canada Research Chairs programme and was done while the second author was on sabbatical at the MADALGO Center for Massive Data Algorithmics, Department of Computer Science, Aarhus University, Denmark.

[1] Linear-space data structures with a query bound of $O\big((N/B)^{1-1/d} + K/B\big)$ block transfers do exist [6, 9], where d is the dimension.

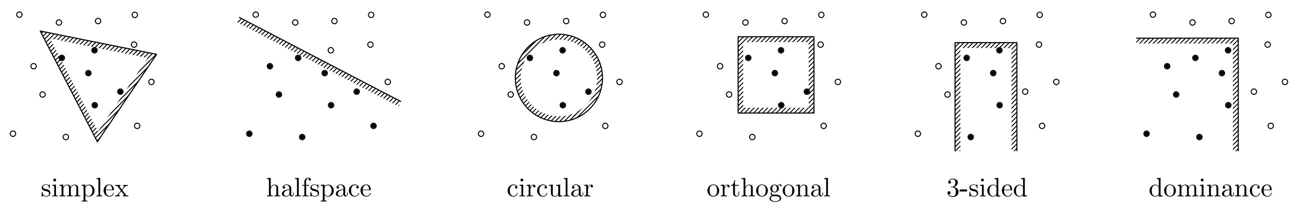

Figure 1: The different query types shown in two dimensions.

solutions to two important special cases of orthogonal range reporting: 2-d *three-sided range reporting* considers query boxes with one side fixed at infinity; 3-d *dominance reporting* considers query boxes whose "bottom-left" vertex is the point $(-\infty, -\infty, -\infty)$. Both problems have been studied extensively, as their solutions can be used as building blocks for general orthogonal range reporting data structures. Here we provide a new cache-oblivious 3-d dominance reporting data structure and a new lower bound on the size of any query-optimal cache-oblivious 2-d three-sided range reporting data structure. Since 2-d three-sided range reporting reduces to 3-d dominance reporting (see Appendix D), both results apply to both problems.

1.1 Previous Work. Our discussion of previous work focuses mostly on 2-d three-sided range reporting and 3-d dominance reporting. A number of related results are mentioned briefly in Table 1.

In the pointer machine model, a classical result by McCreight provides a linear-space data structure that achieves the optimal query bound of $\mathrm{O}(\log N + K)$ for 2-d three-sided range reporting [18]. Optimal results with the same space and query bounds for 3-d dominance reporting and 3-d halfspace range reporting were obtained much more recently [1, 2, 16].

In the I/O model, Arge *et al.* [10] presented a linear-space data structure that achieves the optimal query bound of $\mathrm{O}(\log_B N + K/B)$ block transfers for 2-d three-sided range reporting; their data structure is an I/O-efficient version of McCreight's data structure. For 3-d dominance reporting, a linear-space data structure by Afshani [1] achieves the optimal query bound of $\mathrm{O}(\log_B N + K/B)$ block transfers. For 3-d halfspace range reporting, Afshani and Chan [2] presented an $\mathrm{O}(N \log^* N)$-space data structure with the optimal query bound of $\mathrm{O}(\log_B N + K/B)$ block transfers.

In contrast to these (nearly) linear-space query-optimal data structures in the pointer machine and I/O models, the size of the best cache-oblivious data structures with the optimal query bound for 2-d three-sided range reporting [5, 6, 8, 11], 3-d dominance reporting [5] and 3-d halfspace range reporting [5] is $\mathrm{O}(N \log N)$, despite many attempts to improve on this bound. Interestingly, at least part of this logarithmic (or almost logarithmic) gap is a genuine phenomenon: Afshani *et al.* [4] proved that any cache-oblivious data structure with the optimal query bound for these problems requires $\Omega(N \log^{\varepsilon} \log N)$ space. While this proves that range reporting is harder in the cache-oblivious model than in the I/O model, it does not answer the question whether the $\mathrm{O}(N \log N)$ space bound is the best possible.

1.2 New Results. In this paper, we make two important steps towards closing the gap between the space upper and lower bounds for query-optimal cache-oblivious range reporting data structures. In Section 2, we present a cache-oblivious data structure for 3-d dominance reporting that uses $\mathrm{O}(N\sqrt{\log N} \log\log N)$ space and achieves the optimal query bound of $\mathrm{O}(\log_B N + K/B)$ block transfers. Using a standard reduction, this implies the same space and query bounds for 2-d three-sided range reporting. In Section 3, we prove an improved lower bound of $\Omega(N \log^{\varepsilon} N)$ space for any cache-oblivious 2-d three-sided range reporting data structure that achieves a query bound of $O(\log_B N + K/B)$ block transfers (our result is in fact slightly stronger and holds also for Las-Vegas randomized data structures; see Section 3). Using the same transformations as in [4], this implies the same lower bound for 3-d dominance reporting and 3-d halfspace range reporting. Table 1 summarizes these results and also mentions some consequences for general 2-d and 3-d orthogonal range reporting.

Similar to previous results for 3-d dominance reporting [1, 5], our upper bound construction is based on shallow cuttings [17]. Using one shallow cutting, it is easy to obtain a linear-size data structure that achieves a query bound of $\mathrm{O}(\log_B N + K/B)$ block transfers, for all queries of output size $K = \Theta(K')$, where K' is fixed. By constructing $\log N$ shallow cuttings, for $K' \in \{2^i : 1 \leq i \leq \log N\}$, we cover all output sizes. Since one shallow cutting uses $\Theta(N)$ space in the worst case, this approach uses $\Theta(N \log N)$ space in the worst case. The need to store $\log N$ shallow cuttings is the main obstacle in obtaining an $\mathrm{o}(N \log N)$-space

Query type	New bound	Best previous bound	References
2-d three-sided and 3-d dominance	$\mathrm{O}(N\sqrt{\log N}\log\log N)$ $\Omega(N\log^{\varepsilon} N)$	$\mathrm{O}(N\log N)$ $\Omega(N\log^{\varepsilon}\log N)$	[5, 6, 8, 11] [4]
2-d orthogonal	$\mathrm{O}(N\log^{3/2}\log\log N)$	$\mathrm{O}(N\log^2 N)$	[5, 6, 8, 11]
3-d orthogonal	$\mathrm{O}(N\log^{7/2}\log\log N)$	$\mathrm{O}(N\log^4 N)$	[5]
3-d halfspace	— $\Omega(N\log^{\varepsilon} N)$	$\mathrm{O}(N\log N)$ $\Omega(N\log^{\varepsilon}\log N)$	[5] [4]

Table 1: A comparison of our new space bounds with previous space bounds for cache-oblivious range reporting data structures with the optimal query bound of $\mathrm{O}(\log_B N + K/B)$ block transfers.

data structure based on this idea. Our contribution is to show how to store the $\log N$ shallow cuttings in $\mathrm{O}(N\sqrt{\log N}\log\log N)$ space while still allowing efficient access to each shallow cutting. A previous method to obtain a space bound of $\mathrm{O}(N\log N/\log\log N)$ is to combine range trees with the linear-space 2-d dominance reporting data structure of Arge and Zeh [11], which results in suboptimal—albeit polylogarithmic—query bounds of $\mathrm{O}(\log_2^{\varepsilon} N + \log_B N + K/B)$ block transfers for 2-d three-sided range reporting and $\mathrm{O}(\log_2^{1+\varepsilon} N + K/B)$ block transfers for 3-d dominance reporting. Our data structure is the first $\mathrm{o}(N\log N/\log\log N)$-space data structure with a polylogarithmic query bound and in fact achieves the optimal query bound.

To prove our new lower bound, we use the general framework of the earlier $\Omega(N\log^{\varepsilon}\log N)$ lower bound by Afshani *et al.* [4]. We define a hard point set by placing clusters of points in the plane and arranging the points in each cluster recursively. Then we construct a set of queries that ensure that, at each level of recursion, the points in at least one cluster are duplicated "a lot" in a query-optimal data structure. By applying this argument to each level of recursion, we prove that a constant fraction of the points are duplicated "a lot". The difference to the construction in [4] is the use of a different point set in combination with a new argument to prove duplication at each level of recursion. This argument is at the same time simpler and more powerful than the argument used in [4] and allows us to increase the duplication from $\Omega(\log^{\varepsilon}\log N)$ to $\Omega(\log^{\varepsilon} N)$. We also argue that this argument is "tight" in the sense that no other point set can guarantee a duplication lower bound greater than $\Omega(\log^{\varepsilon} N)$ using the framework of [4]. In fact, we conjecture that it is possible to construct cache-oblivious $\mathrm{O}(N\log^{\varepsilon} N)$-space data structures with the optimal query bound for 3-d dominance reporting and 2-d three-sided range reporting.

2 3-D Dominance Reporting

In this section, we prove the following result.

THEOREM 2.1. *There exists a cache-oblivious data structure that uses $\mathrm{O}(N\sqrt{\log N}\log\log N)$ space to store a set S of N points in $\mathbb{R}^3$ and supports 3-d dominance reporting queries over S using $\mathrm{O}(\log_B N + K/B)$ block transfers.*

Using standard reductions (see, e.g., [5]), this implies the following corollary.

COROLLARY 2.1. *There exist cache-oblivious data structures that support 2-d three-sided range reporting queries, 2-d orthogonal range reporting queries, and 3-d orthogonal range reporting queries using $\mathrm{O}(\log_B N + K/B)$ block transfers and respectively use $\mathrm{O}(N\sqrt{\log N}\log\log N)$, $\mathrm{O}(N\log^{3/2} N\log\log N)$, and $\mathrm{O}(N\log^{7/2} N\log\log N)$ space to store N points.*

As many earlier range searching data structures [13, 17, 19], our 3-d dominance reporting data structure is based on shallow cuttings. In the context of 3-d dominance reporting, a *shallow K-cutting* is a collection of $\mathrm{O}(N/K)$ 3-d dominance query ranges, called *cells*, such that each contains $\mathrm{O}(K)$ points and, for every 3-d dominance query q containing at most K points, there exists a cell completely containing q [1].

The following construction of a 3-d dominance reporting data structure using shallow cuttings is fairly standard: Let $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_t$ be shallow cuttings, where $t := \lceil \log N \rceil$ and $\mathcal{C}_i$ is a shallow 2^i-cutting for the point set S. For every $0 \le i \le t$ and every cell $C \in \mathcal{C}_i$, we store the conflict list Δ_C of C, where the conflict list Δ_R of a region R is the set of points in S contained in R. To answer a 3-d dominance reporting query q, we find a 2-approximation K' of the output size K, locate a cell $C \in \mathcal{C}_i$ that contains q, where $i := \lceil \log K' \rceil$, and finally inspect all points in Δ_C and report those that are contained in q. The approximation K' of the output size can be obtained using $\mathrm{O}(\log_B(N/K))$ block transfers using the cache-oblivious $(1+\varepsilon)$-approximate 3-d dominance counting data structure of [5]. Finding a cell $C \in \mathcal{C}_i$ containing the query q reduces to point location in a planar straight-line subdivision of size $\mathrm{O}(N/2^i) = \mathrm{O}(N/K)$ [1, 16]. Thus, the cell C can be found using $\mathrm{O}(\log_B(N/K))$ block transfers if we store this subdivision using the cache-oblivious planar point location data structure of [12]. Finally, assuming the points in the conflict list of each cell of $\mathcal{C}_i$ are stored consecutively, reporting the points in $q \cap S$ by scanning the points in Δ_C takes $\mathrm{O}(1 + |\Delta_C|/B) = \mathrm{O}(1 + K/B)$ block transfers, and the total query bound is $\mathrm{O}(\log_B N + K/B)$ block transfers.

The approximate dominance counting data structure of [5] uses linear space, as do the point location data structures for the shallow cuttings $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_t$. Storing the conflict lists of the shallow cutting cells naively, however, uses $\mathrm{O}(N \log N)$ space. Here we show how to store them in $\mathrm{O}(N\sqrt{\log N}\log\log N)$ space, while still allowing each conflict list Δ_C to be retrieved using $\mathrm{O}(\log_B N + |\Delta_C|/B)$ block transfers. Since the query procedure inspects one shallow cutting cell, this proves Theorem 2.1.

A simple idea to reduce the space needed to store the conflict lists is to exploit the nesting of cells: For a set of cells $C_1 \supseteq C_2 \supseteq \cdots \supseteq C_k$ (see Figure 2(a)), we can store the points of each conflict list Δ_{C_i} consecutively without duplication by storing the point lists $\Delta_{C_k}, \Delta_{C_{k-1}} \setminus \Delta_{C_k}, \ldots, \Delta_{C_1} \setminus \Delta_{C_2}$ consecutively. By itself, this idea is not very useful, as it can be shown that, in the worst case, most pairs of non-disjoint cells are not nested. In order to make this strategy effective, we use a second idea: we cut each cell into a constant number of boxes. This requires us to assemble the conflict list of a shallow cutting cell from the conflict lists of $\mathrm{O}(1)$ boxes, but one may hope that the boxes have much better nesting properties if the partition into boxes is chosen appropriately. Our data structure is based on these two ideas but is slightly more involved, as we do not know how to guarantee good nesting properties for *all* the boxes. Instead, our data structure can be seen as partitioning the boxes into two sets: one with good nesting properties, while the other one can be stored in space-efficient data structures that allow the conflict list of a query box to be retrieved efficiently.

Let $\mathcal{C}^* := \mathcal{C}_0 \cup \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_t$ be the collection of all shallow cutting cells. We partition $\mathcal{C}^*$ into two subsets $\mathcal{S}$ (sample) and $\mathcal{R}$ (remainder). The cells in $\mathcal{S}$ contain a small number of points ($\mathrm{o}(N \log N)$); thus, we can afford to build linear-size data structures for each cell in $\mathcal{S}$. To store the cells in $\mathcal{R}$, we "clip" each cell C in $\mathcal{R}$ using cells in $\mathcal{S}$; see Figure 2(e). The points in the intersection between C and the cells in $\mathcal{S}$ used to clip C are reported by using the data structures built on these cells in $\mathcal{S}$. The remaining portion of C, called the "tip" of C in what follows, needs to be stored explicitly. As it turns out, we can ensure good nesting properties for almost all tips, which allows the tips to be stored in $\mathrm{o}(N \log N)$ space. By combining this with the $\mathrm{o}(N \log N)$ space required to store the cells in $\mathcal{S}$, we obtain an $\mathrm{o}(N \log N)$ space data structure to store the cells in $\mathcal{C}^*$.

We divide the detailed discussion of our data structure into two parts. In Section 2.1, we define the sets $\mathcal{S}$ and $\mathcal{R}$ and construct data structures on them that allow the conflict list of each shallow cutting cell $C \in \mathcal{C}^*$ to be retrieved using $\mathrm{O}(\log_B N + |\Delta_C|/B)$ block transfers. In Section 2.2, we show that, with the right choice of parameters in the construction of $\mathcal{S}$ and $\mathcal{R}$, the size of the resulting data structure is $\mathrm{O}(N\sqrt{\log N}\log\log N)$.

2.1 Data Structure. Let $0 < p < 1$ be a parameter to be chosen later to minimize the size of the data structure. We obtain the set $\mathcal{S}$ by sampling each element of $\mathcal{C}^*$ independently at random with probability p; the set $\mathcal{R}$ is defined as $\mathcal{R} := \mathcal{C}^* \setminus \mathcal{S}$. For each cell $C \in \mathcal{S}$, we build three 2-d dominance reporting data structures on the projections of the points in Δ_C on the xy-, xz-, and yz-planes, respectively. To discuss the way we represent the cells in $\mathcal{R}$, we need to introduce some notation.

Every cell $C \in \mathcal{C}^*$ corresponds to a dominance query with a query point (x, y, z), which we call the *apex* of C. We say a cell C_1 with apex (x_1, y_1, z_1) *x-clips* another cell C_2 with apex (x_2, y_2, z_2) if $x_1 < x_2$, $y_1 > y_2$, and $z_1 > z_2$. We define the terms "*y-clips*" and "*z-clips*" analogously. See Figures 2(b)–2(d). Note that, if neither C_1 nor C_2 clips the other, one is contained in the other. For a partition of $\mathcal{C}^*$ into two sets $\mathcal{S}$ and $\mathcal{R}$, we say a cell $C_1 \in \mathcal{S}$ *maximally* x-clips a cell $C_2 \in \mathcal{R}$ if C_1 x-clips C_2 and there is no cell in $\mathcal{S}$ that x-clips C_2 and has a larger intersection with C_2 than C_1. In this case, we denote C_1 by $\mu_x(C_2)$. The cells $\mu_y(C_2)$ and $\mu_z(C_2)$ that maximally y- and z-clip C_2 are defined analogously, if they exist. The *tip* of a cell $C \in \mathcal{R}$ is defined as $\hat{C} := C \setminus (\mu_x(C) \cup \mu_y(C) \cup \mu_z(C))$. This is illustrated in Figure 2(e).

In our data structure, every cell $C \in \mathcal{S}$ stores pointers to its 2-d dominance data structures, while every cell $C \in \mathcal{R}$ stores pointers to $\mu_x(C)$, $\mu_y(C)$, and $\mu_z(C)$, as well as to a place in memory where the conflict list of its tip $\hat{C}$ is stored. Retrieving the conflict list of a cell $C \in \mathcal{S}$ reduces to a (degenerate) 2-d dominance query on one of the 2-d dominance reporting data structures of C. Using the 2-d dominance reporting data structure of [11], this takes $\mathrm{O}(\log_B N + |\Delta_C|/B)$ block transfers. For a cell $C \in \mathcal{R}$, we retrieve the

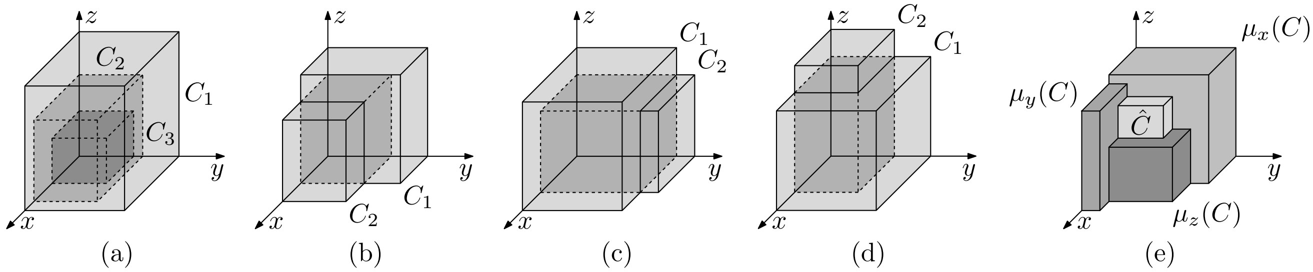

Figure 2: (a) Three nested cells. (b) C_1 x-clips C_2. (c) C_1 y-clips C_2. (d) C_1 z-clips C_2. (e) The tip of a cell C and the cells that maximally clip C in the three dimensions.

conflict lists of $C \cap \mu_x(C)$, $C \cap \mu_y(C)$, $C \cap \mu_z(C)$, and $\hat{C}$ and ensure that we report every point only once (it is easy to test, e.g., whether a point in $C \cap \mu_y(C)$ is also contained in the box $C \cap \mu_x(C)$ and, thus, has already been reported and should not be reported again). Retrieving the points in $C \cap \mu_x(C)$, $C \cap \mu_y(C)$, and $C \cap \mu_z(C)$ reduces to answering 2-d dominance reporting queries on the yz-projection of $\Delta_{\mu_x(C)}$, the xz-projection of $\Delta_{\mu_y(C)}$, and the xy-projection of $\Delta_{\mu_z(C)}$ and, thus, takes $\mathrm{O}(\log_B N + |\Delta_C|/B)$ block transfers using the 2-d dominance reporting data structures of $\mu_x(C)$, $\mu_y(C)$, and $\mu_z(C)$. The points in $\Delta_{\hat{C}}$ can be retrieved using $\mathrm{O}(1+|\Delta_{\hat{C}}|/B)$ block transfers if we store these points consecutively. Our goal, therefore, is to store the conflict lists of all tips space-efficiently while storing the points in each tip's conflict list consecutively.

Let $\hat{\mathcal{R}} := \{\hat{C} : C \in \mathcal{R}\}$ be the set of all tips of cells in $\mathcal{R}$. Our approach for storing the tips in $\hat{\mathcal{R}}$ makes use of a directed graph G, which we call the *clipping graph* of $\hat{\mathcal{R}}$. The vertex set of G is the set of tips in $\hat{\mathcal{R}}$. (Slightly abusing notation, we do not distinguish between a vertex in G and its corresponding tip.) There is an edge from $\hat{C}_1$ to $\hat{C}_2$ in G if and only if C_1 clips C_2 and $\hat{C}_1 \cap \hat{C}_2 \neq \emptyset$. The *weight* of a vertex $\hat{C}$ is the number of points in $\Delta_{\hat{C}}$. The following lemma shows how the clipping graph captures the nesting properties of the tips.

Lemma 2.1. *If there is no edge between $\hat{C}_1$ and $\hat{C}_2$ in G, then $\hat{C}_1 \cap \hat{C}_2 = \emptyset$, $\hat{C}_1 \subseteq \hat{C}_2$ or $\hat{C}_2 \subseteq \hat{C}_1$.*

Proof. Assume $\hat{C}_1$ and $\hat{C}_2$ are non-disjoint, but there is no edge between $\hat{C}_1$ and $\hat{C}_2$ in G. This implies that neither C_1 nor C_2 clips the other. Thus, w.l.o.g. $C_1 \subseteq C_2$. We show that this implies that $\hat{C}_1 \subseteq \hat{C}_2$. Assume the contrary. Then w.l.o.g. the x-ranges of $\hat{C}_1$ and $\hat{C}_2$ are non-disjoint but neither contains the other. If $\mu_x(C_2) \cap \hat{C}_1 = \emptyset$, the x-range of $\hat{C}_2$ contains the x-range of $\hat{C}_1$; if $\mu_x(C_2)$ contains $\hat{C}_1$, then $\hat{C}_1$ and $\hat{C}_2$ are disjoint. Thus, $\mu_x(C_2)$ intersects $\hat{C}_1$ but does not contain it. Since $C_1 \subseteq C_2$ and $\mu_x(C_2)$ x-clips C_2, this implies that $\mu_x(C_2)$ also x-clips C_1, which in turn implies that $\mu_x(C_2) \cap C_1 \subseteq \mu_x(C_1) \cap C_1$. Since, however, $\mu_x(C_1) \cap C_1$ and $\hat{C}_1$ are disjoint, this implies that $\mu_x(C_2) \cap C_1$ and $\hat{C}_1$ are also disjoint, that is, $\mu_x(C_2)$ and $\hat{C}_1$ are disjoint, a contradiction. □

To store the tips in $\hat{\mathcal{R}}$, our goal is to partition $\hat{\mathcal{R}}$ into a small number of subsets that can each be stored in linear space. We obtain this partition using a partial t-colouring of G. A *t-colouring* of a graph G assigns one of t colours to each vertex of G so that no two adjacent vertices receive the same colour. A *partial t-colouring* of G is a t-colouring of a subgraph H of G; the vertices in $V(G) \setminus V(H)$ do not receive any colour. For a partial t-colouring of G, let V_0 be the set of vertices of G that are not assigned a colour, and let $w(V_0)$ be the total weight of these vertices.

Lemma 2.2. *Given a partial t-colouring of the clipping graph G of $\hat{\mathcal{R}}$ that leaves a set V_0 of vertices uncoloured, the conflict lists of the tips in $\hat{\mathcal{R}}$ can be stored in $tN+w(V_0)$ space while storing the points in each conflict list consecutively.*

Proof. Storing the conflict list of each tip in V_0 explicitly uses $w(V_0)$ space for all tips in V_0. The tips of a given colour form an independent set in G. Thus, it suffices to show that the conflict lists of an independent set I in G can be stored in N space. By Lemma 2.1, each pair of tips in I is either disjoint or one is contained in the other. Thus, if we add a box B_0 that contains all tips in I, as well as all points in S, the nesting of the resulting set of boxes $I' := I \cup \{B_0\}$ defines a tree T where $\hat{C}_1$ is the parent of $\hat{C}_2$ if $\hat{C}_2 \subseteq \hat{C}_1$ and there is no box $\hat{C}_3$ in I' with $\hat{C}_2 \subseteq \hat{C}_3 \subseteq \hat{C}_1$. We associate each point in S with the smallest box in I' (node in T) that contains it. Then we order the nodes of T in postorder and store the sets of points associated with the nodes in this order. Clearly, each point is stored only once, and the points contained in each box $\hat{C}$ in I' are stored consecutively because this is exactly the set of points associated with the descendants of $\hat{C}$ in T, inclusive. $\square$

2.2 Space Analysis. We have already argued that the data structure described in Section 2.1 allows the retrieval of the conflict list of a shallow cutting cell C using $O(\log_B N + |\Delta_C|/B)$ block transfers. Thus, to prove Theorem 2.1, it suffices to prove that, with the right choice of the parameter p, the data structure uses $O(N\sqrt{\log N}\log\log N)$ space.

First consider the cells in $\mathcal{S}$. The 2-d dominance reporting data structure of [11] uses linear space. Thus, storing three such data structures per cell in $\mathcal{S}$ uses $O(pN\log N)$ expected space, as the cells in $\mathcal{C}^*$ contain $O(N\log N)$ points in total and each cell is included in $\mathcal{S}$ with probability p.

Next we show how to obtain a partial t-colouring of the clipping graph G of $\hat{\mathcal{R}}$ with $t := (12/p)\log^2\log N$ and such that the expected weight of the uncoloured vertices is $O(N)$. By Lemma 2.2, this implies that the tips in $\hat{\mathcal{R}}$ can be stored in $O((N/p)\log^2\log N)$ expected space. By summing the space bounds for storing the cells in $\mathcal{S}$ and the tips in $\hat{\mathcal{R}}$, we obtain that our data structure uses $O(pN\log N + (N/p)\log^2\log N)$ expected space. For $p := \log\log N/\sqrt{\log N}$, this gives an expected space bound of $O(N\sqrt{\log N}\log\log N)$. In particular, there exists a partition of $\mathcal{C}^*$ into subsets $\mathcal{S}$ and $\mathcal{R}$ for which our data structure uses $O(N\sqrt{\log N}\log\log N)$ space. This proves Theorem 2.1.

To obtain the desired partial t-colouring of G, we first remove all vertices of in-degree greater than $d := (3/p)\log\log N$ from G and add them to V_0. Let G_0 be the subgraph of G induced by the remaining vertices. We repeat the following t times: Let G_{j-1} be the input graph of the jth iteration. We construct a subset I'_j of the vertices of G_{j-1} by sampling each vertex with probability $1/2d$ and define I_j to be the subset of vertices in I'_j that have no in-neighbours in I'_j. We assign colour j to the vertices in I_j and define G_j to be the subgraph of G_{j-1} obtained by removing the vertices in I_j. After the last iteration, we add all vertices of G_t to V_0. This procedure produces a partial t-colouring of G because each set I_j is an independent set. It remains to bound the expected weight of V_0.

Lemma 2.3. *The expected weight of the vertices in V_0 is $O(N)$.*

Proof. The vertices in V_0 can be divided into two subsets: the vertices of in-degree greater than d in G and the vertices of G_t. We show that the expected weight of each of these two sets is $O(N)$.

To bound the weight of all vertices of in-degree greater than d, it suffices to prove that the probability that a vertex of G has in-degree greater than d is at most $3/\log N$. Indeed, since the total weight of all vertices in G is $O(N\log N)$, this implies that the expected weight of the vertices of in-degree greater than d is $O(N)$. So consider a cell $C \in \mathcal{R}$, and let $\mathcal{C}^*_x(C)$, $\mathcal{C}^*_y(C)$, and $\mathcal{C}^*_z(C)$ be the lists of cells in $\mathcal{C}^*$ that x-clip, y-clip, and z-clip C, respectively. Assume further that the cells in these lists are sorted by decreasing x-, y-, and z-coordinates of their apexes, respectively. Let i_x, i_y, and i_z be the positions of $\mu_x(C)$, $\mu_y(C)$, and $\mu_z(C)$ in $\mathcal{C}^*_x(C)$, $\mathcal{C}^*_y(C)$, and $\mathcal{C}^*_z(C)$, respectively. For $\hat{C}$ to have in-degree greater than d in G, we must have $i_x > k$, $i_y > k$ or $i_z > k$, for $k := d/3 = \log\log N/p$. By the definition of $\mu_x(C)$, we have $i_x > k$ if and only if none of the first k elements of $\mathcal{C}^*_x(C)$ is included in $\mathcal{S}$, which happens with probability $(1-p)^k$. The same holds for i_y and i_z. Thus, the probability that $\hat{C}$ has in-degree greater than d in G is at most $3(1-p)^k \le 3/\log N$.

To bound the weight of the vertices in G_t, we prove that $E[w(G_j) \mid w(G_{j-1}) = X] \le (1 - 1/4d)X$ and, hence, $E[w(G_j)] \le (1-1/4d)E[w(G_{j-1})]$. This implies that $E[w(G_t)] \le (1-1/4d)^t w(G_0) = (1-1/4d)^{4d\log\log n} w(G_0) = O(w(G_0)/\log N)$. Since $w(G_0) = O(N\log N)$, this shows that the expected weight of G_t is $O(N)$.

So assume that $w(G_{j-1}) = X$. The probability that a vertex $x \in G_{j-1}$ belongs to I'_j is $1/2d$. Since x has in-degree at most d, the probability that at least one in-neighbour of x belongs to I'_j is at most $d \cdot 1/2d = 1/2$. Thus, the probability that x belongs to I'_j and none of its in-neighbours belongs to I'_j—that is, the probability that x belongs to I_j—is at least $1/2 \cdot 1/2d = 1/4d$. Since this is true for every vertex of G_{j-1}, the expected weight of I_j is at least $X/4d$, that is, the expected weight of G_j is at most $(1-1/4d)X$, as claimed. $\square$

3 Lower Bounds

The main result in this section is the following.

Theorem 3.1. *Let $f(\cdot,\cdot)$ be a monotonically increasing function,*[2] *and $0 < \delta \le 1/2$ a constant. Any cache-oblivious data structure capable of answering 2-d three-sided range reporting queries using $f(\log_B N, K/B)$ block transfers in the worst case, for every block size $B \le N^{2\delta}$, must use $\Omega(\varepsilon N \log^{\varepsilon} N)$ space in the worst case, where $\varepsilon := 1/(3f(\delta^{-1}, 1))$.*

At the end of this section, we extend this result to other, related problems, as well as to Las-Vegas randomized data structures.

[2] We call a function $f(\cdot,\cdot)$ monotonically increasing if it is monotonically increasing in both its arguments.

To prove Theorem 3.1, we consider a cache-oblivious data structure $\mathcal{D}$ to be a collection of memory cells, each of which stores a point in the point set S. For a given block size B, the data structure decides how to group the cells into blocks of size B. We call the resulting collection of blocks a *B-cover* of S. We say a subset of the blocks in this B-cover *covers* a query q if every point in q belongs to at least one of these blocks. The cost of a query q for block size B is the minimum number of blocks in the B-cover required to cover q. Note that this model ignores the cost to locate a set of blocks covering q and that we do not make any assumptions about how the B-covers for different values of B relate to each other. Thus, our framework can be viewed as applying the indexability model of [15] to a range of different block sizes. The following lemma from [4] shows that it suffices to prove Theorem 3.1 for the case $\delta = 1/2$, that is, for arbitrarily large block sizes $B \leq N$. For the sake of completeness, the proof is included in Appendix A.

LEMMA 3.1. ([4]) *If Theorem 3.1 holds for $\delta = 1/2$, then it holds for any $0 < \delta \leq 1/2$.*

The proof of Theorem 3.1 is divided into two parts. In Section 3.1 we construct a set S_0 of m points and a query set Q_0 over S_0 and prove that any data structure that can answer the queries in Q_0 using $\alpha = O(1)$ block transfers, for an appropriate block size B, has to duplicate at least one point in S_0 at least $m^{1/(2\alpha)}$ times. In Section 3.2 we present arguments based on a recursive application of the result in Section 3.1 that show how to boost the number of points with duplication roughly $m^{1/(2\alpha)}$ to $\Omega(N)$, as long as $m = \Theta(\log N / \log \log N)$. In this recursive construction, $f(\log_B N, K/B)$ will be bounded by $\alpha := f(2, 1) = O(1)$, so we obtain the lower bound stated in Theorem 3.1.

The general framework is the same as in [4], but there are two major differences. First, we use a different point set S_0 in Section 3.1 and apply a simpler and yet more powerful argument to prove the existence of a point in S_0 with high duplication. Second, the argument in Section 3.2 is independent of the specific point set used in Section 3.1. Thus, if the point set in Section 3.1 could be replaced with a point set that has better duplication properties, this would automatically improve our lower bound. As we show in Section 3.4, however, the point set S_0 constructed in Section 3.1 is the worst possible point set as far as enforcing duplication of a single point is concerned. In the same section, we argue that the recursive construction in Section 3.2 also cannot be improved by more than a constant factor. Together, these two observations imply that the lower bound in Theorem 3.1 is the strongest possible bound that can be obtained using the framework used here and in [4].

3.1 Duplicating One Point.

Let $t := 2^k$, for some integer k, and $m := 2t-1$. In this section, we construct a set S_0 of m points and a set Q_0 of three-sided queries over S_0 such that any $(\alpha \log t)$-cover $\mathcal{C}_0$ (i.e., a B-cover with block size $B := \alpha \log t$) that covers every query in Q_0 using at most $\alpha = O(1)$ blocks must duplicate at least one point in S_0 at least $m^{1/(2\alpha)}$ times. The duplication of a point p in $\mathcal{C}_0$ is the number of blocks of $\mathcal{C}_0$ that contain p.

To construct such a point set S_0, let R be a rectangular region into which the points in S_0 are to be placed. We define a perfect binary tree T with t leaves. This tree has $2t-1$ nodes representing the points in S_0, and we do not distinguish between nodes of T and points in S_0. We distribute the leaves of T evenly along the main diagonal of R. Every non-leaf node of T has the same x-coordinate as its leftmost descendant leaf and the same y-coordinate as its rightmost descendant leaf. This construction is illustrated in Figure 3(a). The query set Q_0 contains one three-sided query q_ℓ per leaf ℓ of T. This query has ℓ as its top-right corner and the same left boundary as R. Observe that this query contains exactly those points in S_0 that are ancestors of ℓ in T, including ℓ itself.

LEMMA 3.2. *Let $\mathcal{C}_0$ be an $(\alpha \log t)$-cover $\mathcal{C}_0$ of S_0 that covers every query in Q_0 using at most α blocks. Then at least one point in S_0 has duplication $m^{1/(2\alpha)}$ in $\mathcal{C}_0$.*

Proof. Assume no point in S_0 has duplication greater than d. We identify a query $q_\ell \in Q_0$ that cannot be covered using less than $\lceil \log t/\beta \rceil$ blocks in $\mathcal{C}_0$, where $\beta := \lceil \log(\alpha d \log t) \rceil$. Since every query in Q_0 can be covered using at most α blocks, this implies that $\alpha \geq \log t/\beta$ and, hence, $d \geq t^{1/\alpha}/(2\alpha \log t)$, which is no less than $m^{1/(2\alpha)}$, as long as t is not too small.

In the following, we call two points in S_0 *neighbours* if there exists a block in $\mathcal{C}_0$ that contains both of them. We construct a set of $\lceil \log t/\beta \rceil$ points $p_0, p_1, \ldots$ in S_0 that appear along the path from a leaf ℓ of T to the root and such that no two of these points are neighbours. Since the query $q_\ell \in Q_0$ contains these points, this query cannot be covered using less than $\lceil \log t/\beta \rceil$ blocks in $\mathcal{C}_0$.

The first point p_0 we choose is the root of T. Given points $p_0, p_1, \ldots, p_i$, we choose the next point p_{i+1} as follows. If the subtree of T with root p_i has height less than β, p_i is the last point we choose. Otherwise, we consider the set of descendants of p_i at distance β from p_i in T. There are $2^\beta \geq \alpha d \log t$ such descendants. Since the at most d blocks containing p_i contain at most $\alpha d \log t$ points, there are at most $\alpha d \log t - 1$

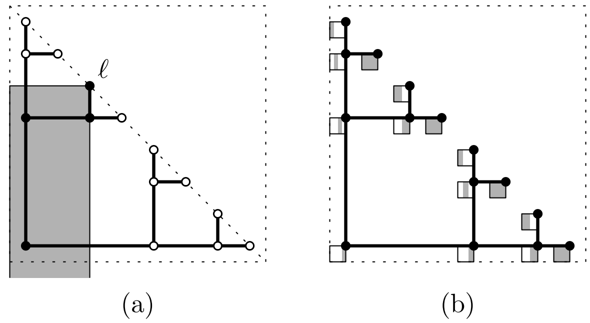

Figure 3: (a) The point set S_0. The tree T is shown as solid edges. The query q_ℓ corresponding to the leaf ℓ is shown in grey. The solid points are contained in q_ℓ, the hollow ones are not. (b) The rectangles R_p associated with the points in S_0 are delimited by solid lines. Their x-disjoint subrectangles R'_p are shown as grey boxes.

neighbours of p_i, and at least one descendant at distance β from p_i must be the root of a subtree of T containing no neighbours of p_i. We choose p_{i+1} to be such a descendant, thereby ensuring that none of the points $p_{i+1}, p_{i+2}, \ldots$ chosen from the subtree with root p_{i+1} is a neighbour of p_i. Since the height of T is $\log t$ and we choose one point p_i for every β levels in T, the number of chosen points is $\lceil \log t/\beta \rceil$, as desired. □

3.2 Duplicating A Constant Fraction of the Points. To prove Theorem 3.1, we apply the argument from the previous section recursively. We define a point set S and a query set Q as follows. We start with a rectangular region R and a number $N_R := N$ of points to be placed into R. For a rectangle R' with $N_{R'}$ points to be placed into R', we distinguish whether $N_{R'} < \sqrt{N}\log N$ or $N_{R'} \geq \sqrt{N}\log N$. If $N_{R'} < \sqrt{N}\log N$, we call R' a *basic rectangle* and place the $N_{R'}$ points arbitrarily into R'. If $N_{R'} \geq \sqrt{N}\log N$, we place a scaled and translated copy of S_0 into R' and apply the same scaling and translation to Q_0 to obtain a query set $Q_{R'}$ over the set of points we place into R'. Next, we associate a rectangular region R_p with every point $p \in S_0$. This rectangle is chosen so that p dominates all points in R_p; see Figure 3(b). Finally, we choose a subrectangle $R'_p \subseteq R_p$, for each point $p \in S_0$, such that these subrectangles are pairwise x-disjoint. We obtain the set of points in R' by replacing each point $p \in S_0$ with a set of $N_{R'_p} := N_{R'}/m$ points placed into R'_p. This set is obtained by applying this procedure recursively to R'_p.

The above construction ensures that every query $q \in Q_{R'}$ contains only points in R'. Indeed, such a query can contain only points in the x-range of R' and the x-disjointness of the subrectangles of each non-basic rectangle ensures that the only points in S contained in the x-range of R' are the points in R' itself.

Next we provide two lemmas that together establish Theorem 3.1. The first one shows that, for every cache-oblivious data structure with a query bound of $f(\log_B N, K/B)$ for the queries in Q, every non-basic rectangle R' has to have a subrectangle R'_p whose points have high average duplication. The second lemma shows that this is enough to prove a high average duplication for a constant fraction of the points in S. Throughout this section, we define $\alpha := f(2,1) = \mathrm{O}(1)$.

Lemma 3.3. *Assume the number m of points in S_0 is at most $\log N$. Then, for every cache-oblivious data structure $\mathcal{D}$ with query bound $f(\log_B N, K/B)$ for the queries in Q, and for every non-basic rectangle R' in the recursive construction of S, there exists a subrectangle R'_p of R' whose points have average duplication at least $m^{1/(2\alpha)}/\alpha$ in $\mathcal{D}$.*

Proof. Let $N'' := N_{R'}/m$ be the number of points in each subrectangle of R', and let $B_{R'} := N'' \log t$ be the output size of each query in $Q_{R'} \subseteq Q$. Since $N_{R'} \geq \sqrt{N}\log N$ and $m \leq \log N$, we have $B_{R'} \geq \sqrt{N}$ and $B_{R'} \geq K$, for every query in $Q_{R'}$. Thus, the $B_{R'}$-cover $\mathcal{C}$ defined by $\mathcal{D}$ must cover every query $q \in Q_{R'}$ using at most $f(\log_{B_{R'}} N, K/B_{R'}) \leq f(2,1) = \alpha$ blocks. We prove that there exists a subrectangle R'_p of R' whose points have average duplication at least $m^{1/(2\alpha)}/\alpha$ in $\mathcal{C}$. Since the number of blocks in $\mathcal{C}$ containing a point $r \in S$ is a lower bound on the number of times r is stored in $\mathcal{D}$, this implies that the average duplication of the points in R'_p in $\mathcal{D}$ is also at least $m^{1/(2\alpha)}/\alpha$.

Our strategy is to transform $\mathcal{C}$ into an $(\alpha \log t)$-cover $\mathcal{C}_0$ of S_0 that covers every query in Q_0 using at most α blocks and such that the duplication of any point $p \in S_0$ in $\mathcal{C}_0$ is at most α times higher than the average duplication of the points in R'_p in $\mathcal{C}$. By Lemma 3.2, there exists a point $p \in S_0$ that is duplicated at least $m^{1/(2\alpha)}$ times in $\mathcal{C}_0$. Thus, the average duplication of the points in the corresponding rectangle R'_p in $\mathcal{C}$ is at least $m^{1/(2\alpha)}/\alpha$.

We construct $\mathcal{C}_0$ as follows. For every block $X' \in \mathcal{C}$, we construct a block $X \in \mathcal{C}_0$ by including the point p in X if and only if X' contains at least N''/α points from R'_p. Since $|X'| = B_{R'} = N'' \log t$, there can be at most $\alpha \log t$ rectangles R'_p contributing at least N''/α points to X' and, hence, X contains at most $\alpha \log t$ points. Thus, $\mathcal{C}_0$ is an $(\alpha \log t)$-cover of S_0.

Next we show that every query $q_\ell \in Q_0$ can be covered using at most α blocks in $\mathcal{C}_0$. Let q'_ℓ be the query in $Q_{R'}$ corresponding to q_ℓ, let $X'_1, X'_2, \ldots, X'_k$, $k \leq \alpha$, be a set of blocks in $\mathcal{C}$ that cover q'_ℓ, and let $X_1, X_2, \dots, X_k$ be the corresponding blocks in $\mathcal{C}_0$. Every rectangle R'_p contained in q'_ℓ contributes N'' points to the output of q'_ℓ. Since these points are contained in $X'_1 \cup X'_2 \cup \dots \cup X'_k$, there exists a block X'_i containing at least $N''/k \geq N''/\alpha$ points from R'_p. The corresponding block $X_i \in \mathcal{C}_0$ contains the point $p \in S_0$. Since a point $p \in S_0$ belongs to q_ℓ if and only if the rectangle R'_p is contained in q'_ℓ, this shows that the blocks $X_1, X_2, \dots, X_k$ cover q_ℓ.

Finally, consider the average duplication $\bar{d}_p$ of the points in a subrectangle R'_p in $\mathcal{C}$. This number is $\bar{d}_p := D_p/N''$, where $D_p := \sum_{r \in R_p} d'_r$ and d'_r is the duplication of point $r \in S$ in $\mathcal{C}$. If a block $X \in \mathcal{C}_0$ contains the point p, its corresponding block $X' \in \mathcal{C}$ contains at least N''/α points of R'_p. Hence, the duplication of p in $\mathcal{C}_0$ is $d_p := |\{X' \in \mathcal{C} : |X' \cap R'_p| \geq N''/\alpha\}|$. Since we can rewrite D_p as $D_p = \sum_{X' \in \mathcal{C}} |X' \cap R'_p|$, we have $d_p \leq \lfloor D_p/(N''/\alpha) \rfloor \leq \alpha \bar{d}_p$. □

By applying Lemma 3.3 to every non-basic rectangle R' in the construction of S, we can ensure that $\Omega(N)$ points in S have average duplication at least $m^{1/(2\alpha)}/\alpha$ in $\mathcal{D}$ and, hence, that $\mathcal{D}$ has size $\Omega(Nm^{1/(2\alpha)}/\alpha)$, as long as the recursion is deep enough. The following lemma states this formally. Its proof is nearly identical to the proof of a similar lemma in [4] and is given in Appendix B.

LEMMA 3.4. *Assume the parameter m in the construction of S is at most $\log N/(3 \log\log N)$, let $\mathcal{D}$ be a data structure storing the points in S, and assume every non-basic rectangle R' in the construction of S has a subrectangle whose points have average duplication at least d in $\mathcal{D}$. Then $\mathcal{D}$ has size $\Omega(dN)$.*

If we choose the parameter t in the definition of S_0 to be the largest power of 2 no greater than $\log N/(6 \log\log N)$, we have $\log N/(6 \log\log N) \leq m < \log N/(3 \log\log N)$. By Lemma 3.3, this shows that every non-basic rectangle R' has a subrectangle whose points have average duplication $\Omega((\log N/(6 \log\log N))^{1/(2\alpha)}/\alpha) = \Omega(\log^{1/(3\alpha)} N/\alpha)$. By Lemma 3.4, this implies that the size of any data structure with query bound $f(\log_B N, K/B)$ for the queries in Q requires $\Omega((N/\alpha)\log^{1/(3\alpha)} N)$ space. This proves Theorem 3.1.

3.3 Extensions. The following two results extend Theorem 3.1 to randomized data structures and to related problems. The proofs are given in Appendices C and D, as they are similar to proofs given in [3].

THEOREM 3.2. *Let $f(\cdot,\cdot)$ be a monotonically increasing function, and $0 < \delta \leq 1/2$ a constant. Any cache-oblivious data structure constructed by a randomized algorithm and capable of answering 2-d three-sided range reporting queries using $f(\log_B N, K/B)$ expected block transfers, for every block size $B \leq N^{2\delta}$, must use $\Omega(\varepsilon N \log^\varepsilon N)$ expected space, where $\varepsilon := 1/(41 f(\delta^{-1}, 1))$.*

COROLLARY 3.1. *Let $f(\cdot,\cdot)$ be a monotonically increasing function, and $0 < \delta \leq 1/2$ a constant. Any cache-oblivious 3-d dominance reporting or 3-d halfspace range reporting data structure constructed by a randomized algorithm and with expected query bound $f(\log_B N, K/B)$, for every block size $B \leq N^{2\delta}$, must use $\Omega(\varepsilon N \log^\varepsilon N)$ expected space, where $\varepsilon := 1/(41 f(\delta^{-1}, 1))$.*

3.4 Tightness of the Lower Bound. In this section, we argue briefly that the result in Theorem 3.1 is the best possible using the general framework used here and in [4], up to the dependence of ε on $f(\cdot,\cdot)$. Thus, if our space lower bound of $\Omega(N \log^\varepsilon N)$ is not tight, a completely different approach is needed to prove a stronger lower bound. We conjecture in fact that the lower bound is tight.

To show that the result in Theorem 3.1 is best possible, recall that the proof relies on first showing that we can ensure a high duplication for at least one point in a point set of size m and then boosting the number of duplicated points to a constant fraction using a recursive construction that replaces every point with N/m points.

The proof of Lemma 3.4 in Appendix B reveals that a recursion depth of at least m is required for this boosting strategy to succeed. Since the recursion depth is at most $\log_m N$, this places an upper bound of roughly $\log N/\log\log N$ on m. As we argue next, we also cannot prove a duplication bound greater than m^ε for any point in the point set, that is, we cannot show a duplication higher than $\log^\varepsilon N$ for a constant fraction of the points. To prove this, we show how to convert any data structure $\mathcal{D}$ of size $\mathrm{O}(N \log^c N)$, for some constant c, into a data structure that duplicates every point at most N^ε times. The resulting data structure has the same size as the original data structure, and its query bound is only $\mathrm{O}(1/\varepsilon)$ times higher than for the original data structure. Thus, for constant ε, the space and query bounds change by only a constant factor, while reducing the duplication of every point to at most N^ε.

Since $\mathcal{D}$ has size $\mathrm{O}(N \log^c N)$, there are $\mathrm{O}(N^{1-\varepsilon} \log^c N)$ points that are duplicated more than N^ε times. We remove these points from the data structure and store them in a new data structure of the same type as $\mathcal{D}$. This data structure has size $\mathrm{O}(N^{1-\varepsilon} \log^{2c} N)$ and, hence, contains at most $\mathrm{O}(N^{1-2\varepsilon} \log^{2c} N)$ points that are duplicated more than

N^ε times. Again, we remove these points and store them in a new data structure. After repeating this $1/\varepsilon$ times, we are left with $1/\varepsilon$ data structures of total size $O(N \log^c N)$ and each with the same query bound as the original data structure $\mathcal{D}$. Every point is stored in only one data structure and is stored at most N^ε times in this data structure. Since range reporting is decomposable, we can now answer range queries by querying each data structure in turn, which takes at most $1/\varepsilon$ times longer than using $\mathcal{D}$.

References

[1] P. Afshani. On dominance reporting in 3D. In *Proceedings of the 16^th European Symposium on Algorithms*, volume 5193 of *Lecture Notes in Computer Science*, pages 41–51. Springer-Verlag, 2008.

[2] P. Afshani and T. M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the 20^th ACM-SIAM Symposium on Discrete Algorithms*, pages 180–186, 2009.

[3] P. Afshani, C. Hamilton, and N. Zeh. Cache-oblivious range reporting with optimal queries requires superlinear space. *Discrete and Computational Geometry*. To appear.

[4] P. Afshani, C. Hamilton, and N. Zeh. Cache-oblivious range reporting with optimal queries requires superlinear space. In *Proceedings of the 25^th ACM Symposium on Computational Geometry*, pages 277–286, 2009.

[5] P. Afshani, C. Hamilton, and N. Zeh. A general approach for cache-oblivious range reporting and approximate range counting. *Computational Geometry: Theory and Applications*, 43(8):700–712, 2010.

[6] P. K. Agarwal, L. Arge, A. Danner, and B. Holland-Minkley. Cache-oblivious data structures for orthogonal range searching. In *Proceedings of the 19^th ACM Symposium on Computational Geometry*, pages 237–245, 2003.

[7] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[8] L. Arge, G. S. Brodal, R. Fagerberg, and M. Laustsen. Cache-oblivious planar orthogonal range searching and counting. In *Proceedings of the 21^st ACM Symposium on Computational Geometry*, pages 160–169, 2005.

[9] L. Arge, M. de Berg, and H. J. Haverkort. Cache-oblivious R-trees. In *Proceedings of the 21^st ACM Symposium on Computational Geometry*, pages 170–179, 2005.

[10] L. Arge, V. Samoladas, and J. S. Vitter. On two-dimensional indexability and optimal range search indexing. In *Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 346–357, 1999.

[11] L. Arge and N. Zeh. Simple and semi-dynamic structures for cache-oblivious planar orthogonal range searching. In *Proceedings of the 22^nd ACM Symposium on Computational Geometry*, pages 158–166, 2006.

[12] M. A. Bender, R. Cole, and R. Raman. Exponential structures for efficient cache-oblivious algorithms. In *Proceedings of the 29^th International Colloquium on Automata, Languages and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 195–207. Springer-Verlag, 2002.

[13] T. M. Chan. Random sampling, halfspace range reporting, and construction of ($\leq k$)-levels in three dimensions. *SIAM Journal on Computing*, 30:561–575, 2000.

[14] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40^th IEEE Symposium on Foundations of Computer Science*, pages 285–297, 1999.

[15] J. M. Hellerstein, E. Koutsoupias, and C. H. Papadimitriou. On the analysis of indexing schemes. In *Proceedings of the 16^th ACM Symposium on Principles of Database Systems*, pages 249–256, 1997.

[16] C. Makris and A. Tsakalidis. Algorithms for three-dimensional dominance searching in linear space. *Information Processing Letters*, 66(6):277–283, 1998.

[17] J. Matoušek. Reporting points in halfspaces. *Computational Geometry: Theory and Applications*, 2(3):169–186, 1992.

[18] E. M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985.

[19] E. A. Ramos. On range reporting, ray shooting, and k-level construction. In *Proceedings of the 15^th ACM Symposium on Computational Geometry*, pages 390–399, 1999.

A Proof of Lemma 3.1

Consider a particular choice of N, f, and δ in Theorem 3.1, and let $N' := N^{2\delta}$ and $f'(x, y) := f(x/(2\delta), y)$. Since we assume Theorem 3.1 holds for $\delta = 1/2$, there exists a point set S' of size N' and a query set Q' over S' such that any data structure achieving a query bound of $f'(\log_B N', K/B)$, for the queries in Q' and all block sizes $B \leq N'$, needs to use $\Omega(\varepsilon N' \log^\varepsilon N')$ space to store S', where $\varepsilon := 1/(3f'(2, 1)) = 1/(3f(\delta^{-1}, 1))$.

Now we construct a point set $S := S_1 \cup S_2 \cup \cdots \cup S_n$, where $n := N/N'$ and each S_i is a translated copy of the point set S'. We choose the translation vectors so that the rightmost point in S_i is to the left of the leftmost point of S_j, for all $1 \leq i < j \leq n$. Next we define a query set $Q := Q_1 \cup Q_2 \cup \cdots \cup Q_n$, where Q_i is a translated copy of the query set Q'; the query $q \in Q_i$ corresponding to a query $q' \in Q'$ contains exactly those points in S_i corresponding to the set of points in S' contained in q'.

Since S_i is a copy of S' and Q_i is a copy of Q', a data structure capable of answering every query in Q_i using $f'(\log_B N', K/B) = f(\log_B(N^{2\delta})/(2\delta), K/B) =$

$f(\log_B N, K/B)$ block transfers, for every $B \leq N' = N^{2\delta}$, needs to use $\Omega(\varepsilon N' \log^{\varepsilon} N') = \Omega(\varepsilon N' \log^{\varepsilon} N)$ space to store the points in S_i. Since this is true for every subset S_i of S, any data structure capable of answering the queries in Q using $f(\log_B N, K/B)$ block transfers, for all $B \leq N^{2\delta}$, needs to use $\Omega(\varepsilon n N' \log^{\varepsilon} N) = \Omega(\varepsilon N \log^{\varepsilon} N)$ space to store S.

B Proof of Lemma 3.4

To bound the size of $\mathcal{D}$ by $\Omega(dN)$, it suffices to construct a set of $\Omega(N)$ points with average duplication d. To construct this set of points, we apply the following recursive selection process, starting with $R' = R$. By the assumption of the lemma, each non-basic rectangle R' has a subrectangle R'_p whose points have average duplication at least d in $\mathcal{D}$. We add the points in R'_p to the set of selected points and recurse on all other subrectangles of R' unless the subrectangles of R' are basic rectangles. This clearly results in a set of selected points with average duplication at least d.

To prove that we select $\Omega(N)$ points, we view this selection process as proceeding in iterations. Initially, no points are selected. In the ith iteration, we consider all rectangles R' at depth i in the recursive construction of S (counting iterations starting with 0 and defining the depth of the top-level rectangle R to be 0) and whose points are not selected. For each such rectangle R', we select a subrectangle R'_p of R' and add the points in R'_p to the set of selected points.

Since every subrectangle of a rectangle R' containing $N_{R'}$ points contains $N_{R'}/m$ points, each iteration selects a $1/m$ fraction of the points not selected before this iteration. Thus, as long as less than $N/2$ points are selected, each iteration selects at least $N/2m$ new points. In particular, after at most m iterations, the number of selected points is at least $N/2$. Next we show that the recursion depth in the construction of the point set S is at least m, thereby allowing for the m iterations of this selection process necessary to select at least $N/2$ points.

Since the recursion stops when each rectangle at the current level of recursion contains at most $\sqrt{N} \log N$ points and every subrectangle of a non-basic rectangle R' containing $N_{R'}$ points contains $N_{R'}/m$ points, the recursion depth in the construction of the point set S is

$$\log_m \frac{N}{\sqrt{N} \log N} \geq \log_m N^{1/3} = \frac{\log N}{3 \log m} \geq \frac{\log N}{3 \log \log N} \geq m,$$

for N sufficiently large and because $m \leq \frac{\log N}{3 \log \log N}$. This completes the proof.

C Extension of the Lower Bound to Las-Vegas Data Structures

While Las-Vegas data structures are not considered in [4], the journal version [3] does show that the lower bound of [4] also holds for Las-Vegas data structures. The argument presented in this section is a combination of the techniques in [3] for dealing with randomization in the data structure and the arguments from Section 3.1 to enforce duplication of a point in S_0.

In a randomized data structure, both the construction algorithm and the query procedure have access to a sequence of random bits. The random bits used during the construction influence the shape of the data structure, while the random bits used by the query procedure influence which blocks are read to answer a given query. In a Las-Vegas data structure, the random bits may influence the costs of individual queries, but the answer provided by a query must always be correct. To prove Theorem 3.2, it suffices to prove it for $\delta = 1/2$. As before, Lemma 3.1 then extends the result to smaller values of δ.

The randomness in the query procedure can be eliminated using the following argument. A randomized query procedure for a data structure $\mathcal{D}$ would make random choices in selecting the blocks to cover a given query q. However, since we ignore the cost of selecting the blocks to cover a query q, we essentially consider an omnipotent query algorithm that always selects the minimum number of blocks in $\mathcal{D}$ to cover q. Randomness in the query procedure cannot reduce this number of blocks for a fixed data structure $\mathcal{D}$.

As a model of randomness in the construction of the data structure, we assume that the construction algorithm uses a finite number, b, of random bits. Depending on the values of these b bits, the algorithm constructs one of $n := 2^b$ data structures $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$. The expected size of the constructed data structure $\mathcal{D}$ is the average size of $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$. A query q has expected cost $f(\log_B N, K/B)$ on $\mathcal{D}$ if, for every block size $B \leq N$, the average number of blocks in the B-covers $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$ defined by $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$ that are needed to cover q is $f(\log_B N, K/B)$.

To prove Theorem 3.2, we consider the point set S and the query set Q constructed in Section 3.2 and mimic the proof of Theorem 3.1. To prove Theorem 3.1, we first showed that every rectangle R' has a subrectangle whose points have average duplication $\Omega(m^{1/(2\alpha)}/\alpha) = \Omega(\log^{1/(3\alpha)} N/\alpha)$ in $\mathcal{D}$. By Lemma 3.4, this implied Theorem 3.1. This proof was based on the fact that *every* query $q \in Q_{R'}$ can be covered with at most α blocks in the $B_{R'}$-cover of S defined by $\mathcal{D}$. For Las-Vegas data structures, we need to argue more carefully because the query bound of such a data structure holds only in the expected sense and, hence, there may be no data structure among $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$ whose $B_{R'}$-cover can cover *every* query in $Q_{R'}$ with at most α blocks. Nevertheless, we can show that, for every non-basic rectangle R' in the recursive construction of S, there are at least $n/2$ data structures among $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$ such that, for each such data structure $\mathcal{D}_i$, the points in some subrectangle of R' have average duplication $\Omega(m^{1/(40\alpha)}/\alpha) = \Omega(\log^{1/(41\alpha)} N/\alpha)$ in $\mathcal{D}_i$. This suffices to show that the total size of the data structures $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$ is $\Omega((nN/\alpha)\log^{1/(41\alpha)} N)$, that is, their average size is $\Omega((N/\alpha)\log^{1/(41\alpha)} N)$.

First we need to extend Lemma 3.2 to *partial* $(\alpha \log t)$-covers of S_0. A partial $(\alpha \log t)$-cover $\mathcal{C}_0$ is a collection of blocks of size $\alpha \log t$ that do not necessarily contain all points of S_0. However, if such a partial $(\alpha \log t)$-cover $\mathcal{C}_0$ covers a query q, then there exists a collection of blocks in $\mathcal{C}_0$ containing all points in q.

Lemma C.1. *For a partial $(\alpha \log t)$-cover $\mathcal{C}_0$ of S_0 that covers at least $3t/4$ of the queries in Q_0 using at most α blocks, at least one point in S_0 has duplication at least $m^{1/(5\alpha)}$ in $\mathcal{C}_0$.*

Proof. Assume no point in S_0 has duplication greater than d in $\mathcal{C}_0$, and let $\beta := \lceil \log(4\alpha d \log t)\rceil$. We show that there exists a query $q_\ell \in Q_0$ that can be covered using at most α blocks but not using less than $\log t/(4\beta)$ blocks. This implies that $\alpha \geq \log t/(4\beta)$ and, hence, $d \geq t^{1/(4\alpha)}/(8\alpha \log t)$, which is no less than $m^{1/(5\alpha)}$, as long as t is not too small.

As in the proof of Lemma 3.2, we call two points in S_0 *neighbours* if there exists a block in $\mathcal{C}_0$ containing both points. Let K_i be the number of points at distances $0, \beta, \ldots, (i-1)\beta$ from the root of the tree T used to define the point set S_0, and let L_i be the number of points at distance $i\beta$ from the root. We have $K_i < 2^{(i-1)\beta+1}$ and $L_i = 2^{i\beta} > 2^{\beta-1}K_i \geq (2\alpha d \log t)K_i$. Since no point has more than $\alpha d \log t$ neighbours, this implies that at least half the points on level $i\beta$ have no neighbours on levels $0, \beta, \ldots, (i-1)\beta$. We call these points on level $i\beta$ *exposed*. Our goal is to show that there exists a root-leaf path in T that corresponds to a query q_ℓ covered by at most α blocks in $\mathcal{C}_0$ and visits at least $\log t/(4\beta)$ exposed points. Since no two exposed points on this path are neighbours, this implies that it takes at least $\log t/(4\beta)$ blocks in $\mathcal{C}_0$ to cover q_ℓ, as claimed.

So consider a random root-leaf path in T. Since at least half the points on each level $i\beta$ are exposed and the height of T is $\log t$, the expected number of exposed points visited by this path is at least $\log t/(2\beta)$. On the other hand, no root-leaf path visits more than $\log t/\beta$ exposed points. This implies that at least $t/3$ root-leaf paths in T visit at least $\log t/(4\beta)$ exposed points. Since at least $3t/4$ queries in Q_0 can be covered using at most α blocks in $\mathcal{C}_0$, this implies that there exists a leaf ℓ of T such that the query q_ℓ can be covered using at most α blocks in $\mathcal{C}_0$ and the path from ℓ to the root of T visits at least $\log t/(4\beta)$ exposed points. This finishes the proof. □

Using Lemma C.1, we can now prove the following equivalent of Lemma 3.3.

Lemma C.2. *For every non-basic rectangle R', there exists a set $\mathfrak{D} \subseteq \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$ of at least $n/2$ data structures such that, for every $\mathcal{D}_i \in \mathfrak{D}$, the points in some subrectangle of R' have average duplication $\Omega(m^{1/(40\alpha)}/\alpha)$ in $\mathcal{D}_i$.*

Proof. As in the proof of Lemma 3.3, let $N'' := N_{R'}/m$ be the number of points in each subrectangle of R', and let $B_{R'} := N'' \log t$ be the output size of every query $q \in Q_{R'}$. Consider the $B_{R'}$-covers $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$ of S defined by $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$. We say a query $q \in Q_{R'}$ is *cheap* for $\mathcal{C}_k$ if q can be covered using at most $8f(\log_{B_{R'}} N, K/B_{R'}) = 8\alpha$ blocks in $\mathcal{C}_k$, and *expensive* otherwise. Since the average number of blocks in $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$ needed to cover a query $q \in Q_{R'}$ is at most α, q is cheap for at least $7n/8$ of the $B_{R'}$-covers $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$.

Next we say a $B_{R'}$-cover $\mathcal{C}_k$ is *good* if at least $3t/4$ queries in $Q_{R'}$ are cheap for $\mathcal{C}_k$. Since each query is cheap for at least $7n/8$ of the $B_{R'}$-covers $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$, at least $n/2$ of these $B_{R'}$-covers are good. We prove that, for each good $B_{R'}$-cover $\mathcal{C}_k$, there exists a subrectangle R'_p of R' whose points have average duplication $\Omega(m^{1/(40\alpha)}/\alpha)$ in $\mathcal{C}_k$ and, hence, in $\mathcal{D}_k$. Since there are $n/2$ good $B_{R'}$-covers, this proves the lemma.

Similar to the proof of Lemma 3.3, our strategy is to turn a good $B_{R'}$-cover $\mathcal{C}_k$ of S into a partial $(8\alpha \log t)$-cover $\mathcal{C}_0$ of S_0 that covers at least $3t/4$ queries in S_0 using at most 8α blocks and such that the duplication of a point p in $\mathcal{C}_0$ is at most 8α times higher than the average duplication of the points in R'_p in $\mathcal{C}_k$. By Lemma C.1, there exists a point $p \in S_0$ that has duplication at least $m^{1/(40\alpha)}$ in $\mathcal{C}_0$ (replacing α with 8α in the lemma). The points in the corresponding rectangle R'_p have average duplication at least $m^{1/(40\alpha)}/(8\alpha) = \Omega(m^{1/(40\alpha)}/\alpha)$ in $\mathcal{C}_k$.

We construct $\mathcal{C}_0$ as follows. For every block $X' \in \mathcal{C}_k$, we construct a block $X \in \mathcal{C}_0$. This block X contains a point $p \in S_0$ if and only if X' contains at least $N''/(8\alpha)$ points from the subrectangle R'_p. The same arguments as in the proof of Lemma 3.3 show that every block $X \in \mathcal{C}_0$ contains at most $8\alpha \log t$ points and that the duplication of a point p in $\mathcal{C}_0$ is at most 8α times higher than the average duplication of the points in R'_p in $\mathcal{C}_k$. Thus, it suffices to prove that $\mathcal{C}_0$ covers at least $3t/4$ queries in Q_0 using at most 8α blocks. In particular, we prove that this is true for every query $q_\ell \in Q_0$ corresponding to a cheap query $q'_\ell \in Q_{R'}$, of which there are at least $3t/4$.

So consider a query $q_\ell \in Q_0$ corresponding to a cheap query $q'_\ell \in Q_{R'}$, let p be a point contained in q_ℓ, let $X'_1, X'_2, \ldots, X'_h$, $h \leq 8\alpha$, be a set of blocks in $\mathcal{C}_k$ that cover q'_ℓ, and let $X_1, X_2, \ldots, X_h$ be the corresponding blocks in $\mathcal{C}_0$. Since R'_p contains N'' points and these points are contained in $X'_1 \cup X'_2 \cup \cdots \cup X'_h$, there exists a block X'_i containing at least $N''/h \geq N''/(8\alpha)$ points from R'_p. The corresponding block X_i contains the point p. Since this is true for every point p contained in q_ℓ, this shows that the set of blocks $X_1, X_2, \ldots, X_h$ covers q_ℓ. $\square$

Using Lemma C.2, we can now prove that the average size of the data structures $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$—that is, the expected size of $\mathcal{D}$—is $\Omega((N/\alpha)m^{1/(40\alpha)}) = \Omega((N/\alpha)\log^{1/(41\alpha)} N)$. To this end, we view each data structure $\mathcal{D}_k$ as storing a separate copy S_k of the point set S, and we consider n copies $R'_1, R'_2, \ldots, R'_n$ of each rectangle R' in the recursive construction of S, one per copy S_k of S. We call a rectangle R'_k *accounted for* if there exists a rectangle $R''_k \supseteq R'_k$ such that the points in R''_k have average duplication $\Omega(m^{1/(40\alpha)}/\alpha)$ in $\mathcal{D}_k$. We call a point $p \in S_k$ accounted for if it is contained in a rectangle R'_k that is accounted for. Thus, the average duplication of all accounted-for points in S_k in $\mathcal{D}_k$ is $\Omega(m^{1/(40\alpha)}/\alpha)$. Our goal is to show that the total number of accounted-for points in $S_1, S_2, \ldots, S_n$ is $\Omega(nN)$, which proves that the total size of the data structures $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$ is $\Omega((nN/\alpha)m^{1/(40\alpha)})$, that is, their average size is $\Omega((N/\alpha)m^{1/(40\alpha)})$, as desired.

Consider all non-basic rectangles R' at depth i in the recursive construction of S that have more than $3n/4$ unaccounted-for copies among $R'_1, R'_2, \ldots, R'_n$, and let N_i be the total number of points in these rectangles R'. By Lemma C.2, at least $n/2$ of the copies $R'_1, R'_2, \ldots, R'_n$ of such a rectangle R' have subrectangles whose points have average duplication $\Omega(m^{1/(40\alpha)}/\alpha)$. Hence, there must be at least $n/4$ copies of R' that are unaccounted for and have an accounted-for subrectangle. Since every subrectangle of R' contains $N_{R'}/m$ points, this shows that there are at least $(n/4) \cdot (N_i/m) = nN_i/(4m)$ points that are accounted for by rectangles at depth $i+1$ but not by rectangles at depth i. Now we consider two cases.

If there exists a recursion depth i such that $N_i \leq N/2$, the boxes at this depth with at least $n/4$ accounted-for copies contain at least $N/2$ points. Thus, the number of accounted-for points in $S_1, S_2, \ldots, S_n$ is at least $(n/4) \cdot (N/2) = nN/8$.

If $N_i > N/2$, for every recursion depth i, every level of recursion introduces $nN_i/(4m) > nN/(8m)$ accounted-for points that were unaccounted for at the previous level. In Appendix B, we showed that the construction of S has at least m levels of recursion. Thus, the total number of accounted-for points is at least $m \cdot nN/(8m) = nN/8$ in this case, as well. Since we obtain a lower bound of $\Omega(nN)$ accounted-for points in both cases, the proof is finished.

D Extension of Lower Bound to Other Range Reporting Problems

The lower bounds of Theorems 3.1 and 3.2 extend to 3-d dominance reporting and 3-d halfspace range reporting using reductions provided in [4]. In particular, we map the point set S and query set Q constructed in Section 3.2 to sets $\phi(S) := \{\phi(p) : p \in S\}$ and $\phi(Q) := \{\phi(q) : q \in Q\}$ such that $\phi(p) \in \phi(q)$ if and only if $p \in q$. Thus, any 3-d dominance reporting or 3-d halfspace range reporting data structure that achieves a query bound of $f(\log_B N, K/B)$ for the queries in $\phi(Q)$ over $\phi(S)$ must obey the space lower bound stated in Theorem 3.1 or 3.2 depending on whether the query bound holds in the worst or expected case. This proves Corollary 3.1. It remains to provide the mapping $\phi(\cdot)$.

3-d dominance reporting. For 3-d dominance reporting, the mapping is obtained straightforwardly using a general reduction that allows any 3-d dominance reporting data structure to be used as a 2-d three-sided range reporting data structure. We map every point $p = (x_p, y_p) \in S$ to the point $\phi(p) := (x_p, -x_p, y_p)$ and every query $q = [l, r] \times (-\infty, y]$ to the query range $\phi(q) := (-\infty, r] \times (-\infty, -l] \times (-\infty, y]$. It is easy to verify that $\phi(p) \in \phi(q)$ if and only if $p \in q$.

3-d halfspace range reporting. There is no general reduction from 2-d three-sided range reporting to 3-d halfspace range reporting. However, a reduction that works for the point set S and query set Q constructed in Section 3.2 is sufficient. The construction we use is essentially identical to the one provided in [4]. We use the fact that there exists a general reduction from 2-d parabolic range reporting to 3-d halfspace range reporting [2]; in 2-d parabolic range reporting, each query range is bounded from above by a parabola of the form $y = a(x - x_p)^2 + y_p$, where $a \leq 0$ and $p = (x_p, y_p)$ is the apex of the query. Thus, it suffices to construct a point set $\phi(S)$ in the plane and a set $\phi(Q)$ of parabolic queries such that $p \in q$ if and only if $\phi(p) \in \phi(q)$, for all $p \in S$ and $q \in Q$. The result from [4] we require here is the following.[3]

LEMMA D.1. *Given two rectangles $R' \subseteq E(R')$, an $m \times n$ grid of subrectangles of R', and a set $Q_{R'}$ of three-sided queries over this set of subrectangles, each subrectangle R'' of R' can be replaced with a subrectangle $\phi(R'') \subseteq R'$ and each query $q \in Q_{R'}$ can be replaced with a parabolic query $\phi(q)$ such that*

(i) The x-ranges of the subrectangles $\phi(R'')$ are disjoint,

(ii) A query $\phi(q)$ either contains a subrectangle $\phi(R'')$ or is disjoint from it,

(iii) A query $\phi(q)$ contains a subrectangle $\phi(R'')$ if and only if q contains R'', and

(iv) Every query $\phi(q)$ intersects only the bottom edge of $E(R')$.

To construct the sets $\phi(S)$ and $\phi(Q)$, where the queries in $\phi(Q)$ are parabolic queries over $\phi(S)$, we follow the recursive construction of S and Q and replace each rectangle R' and its corresponding query set $Q_{R'}$ with a corresponding rectangle $\phi(R')$ and query set $\phi(Q_{R'})$. We start with the top-level rectangle R and define $\phi(R) := R$ and $E(R) := R$. For a non-basic rectangle R' with enclosing rectangle $E(R') \supseteq R'$, we first define the set of subrectangles of R' and the set of queries in $Q_{R'}$ as in Section 3.2. These subrectangles are a subset of the cells of a $t \times m$ grid. We replace each subrectangle R'_p of R' with a rectangle $\phi(R'_p)$ and each query $q \in Q_{R'}$ with a query $\phi(q)$ using Lemma D.1. Next we allocate $N_{R'}/m$ points to each rectangle $\phi(R'_p)$. If $N_{R'}/m < \sqrt{N} \log N$, we place the points allocated to a subrectangle $\phi(R'_p)$ arbitrarily into $\phi(R'_p)$. Otherwise, we recursively apply this construction to each subrectangle $\phi(R'_p)$ with enclosing rectangle $E(\phi(R'_p))$ defined to be the smallest rectangle that contains $\phi(R'_p)$ and touches the bottom boundary of $E(R')$.

Now observe that a query $\phi(q) \in \phi(Q_{R'})$ contains a subrectangle $\phi(R'_p)$ if and only if its corresponding query $q \in Q_{R'}$ contains the subrectangle R'_p. Thus, $\phi(q)$ contains a point $\phi(r) \in \phi(R')$ if and only if q contains the point $r \in R'$.

The other observation we make is that a query $\phi(q) \in \phi(Q_{R'})$ contains no point outside of $\phi(R')$, just as a query $q \in Q_{R'}$ contains only points from R'. Indeed, the x-disjointness of the subrectangles $\phi(R'_p)$ of each rectangle $\phi(R')$ ensures that there are no points of $\phi(S)$ in the x-range of $\phi(R')$ but outside of $\phi(R')$. Every query $\phi(q) \in \phi(Q_{R'})$ leaves the x-range of $E(\phi(R'))$ (and, hence, of $\phi(R')$) only below the bottom boundary of $E(\phi(R'))$ because it only intersects the bottom boundary of $E(\phi(R'))$. Since our construction of the rectangles $E(\phi(R'))$ ensures that their bottom boundaries coincide with the bottom boundary of the top-level rectangle R and there are no points in $\phi(S)$ outside of R, this shows that no query $\phi(q) \in \phi(Q_{R'})$ contains a point not in $\phi(R')$.

Together, these two observations show that $\phi(r) \in \phi(q)$ if and only if $r \in q$, for all $r \in S$ and $q \in Q$; that is, the mapping $\phi(\cdot)$ we have constructed has the desired properties.

[3] This result is not stated as a lemma in [4], but it is exactly what the construction in [4] proves.