

Ordered and Unordered Top-K Range Reporting in Large Data Sets

Peyman Afshani*

Gerth Stølting Brodal†

Norbert Zeh‡

Abstract

We study the following problem: Given an array A storing N real numbers, preprocess it to allow fast reporting of the K smallest elements in the subarray $A[i, j]$ in sorted order, for any triple (i, j, K) with $1 \leq i \leq j \leq N$ and $1 \leq K \leq j - i + 1$. We are interested in scenarios where the array A is large, necessitating an I/O-efficient solution.

For a parameter f with $1 \leq f \leq \log_m n$, we construct a data structure that uses $O((N/f) \log_m n)$ space and achieves a query bound of $O(\log_B N + fK/B)$ I/Os,¹ where B is the block size, M is the size of the main memory, $n := N/B$, and $m := M/B$. Our main contribution is to show that this solution is nearly optimal. To be precise, we show that achieving a query bound of $O(\log^\alpha n + fK/B)$ I/Os, for any constant α , requires $\Omega\left(N \frac{f^{-1} \log_M n}{\log(f^{-1} \log_M n)}\right)$ space, assuming $B = \Omega(\log N)$. For $M \geq B^{1+\epsilon}$, this is within a $\log \log_m n$ factor of the upper bound. The lower bound assumes indivisibility of records and holds even if we assume K is always set to $j - i + 1$.

We also show that it is the requirement that the K smallest elements be reported in sorted order which makes the problem hard. If the K smallest elements in the query range can be reported in any order, then we can obtain a linear-size data structure with a query

bound of $O(\log_B N + K/B)$ I/Os.

1 Introduction

In this paper, we study the following variant of one-dimensional range reporting: Given an array A storing N real numbers, preprocess it to allow fast reporting of the K smallest elements in the array $A[i, j]$, for any triple (i, j, K) with $1 \leq i \leq j \leq N$ and $1 \leq K \leq j - i + 1$. We study two variants of this problem, one where the reported elements have to be reported in sorted order and one where this is not required. We call these variants *ordered* and *unordered top-K range reporting*, respectively.

Our solution to unordered top- K range reporting is required as a building block for the solution to the ordered variant and to demonstrate that it is exactly the ordering requirement that makes the problem hard. Ordered top- K range reporting generalizes and is motivated by the following natural problem in information retrieval and web search engines. Consider a collection of text documents (web pages) stored in a trie or suffix tree to allow identifying all documents containing a query term. A query returns a node of the trie, and the documents corresponding to its descendant leaves are those containing the query term. If the number of matching documents is large, we only want to report the “most relevant” documents according to some ranking and, even if the number of matching documents is fairly small, it is desirable to list matches by decreasing relevance (rank).

Often, a significant part of the rank of a match to a query is independent of the query, such as for example the PageRank [13] of a web page. Thus, as an initial filtering step, it is useful to retrieve only the top K matches with respect to this static rank component. If we number the leaves of the trie left to right and store the i th leaf in position i of an array A , this is exactly an ordered top- K range reporting query with query interval restricted to correspond to the set of descendants of a trie node.

Since search engines and information retrieval applications often deal with massive amounts of data, it is useful to seek an I/O-efficient solution to this problem, that is, one that aims to minimize the number of disk

*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.

†MADALGO Center for Massive Data Algorithmics, Department of Computer Science, Aarhus University, Denmark. Email: gerth@madalgo.au.dk. Research 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 third author was on sabbatical at the MADALGO Center for Massive Data Algorithmics, Department of Computer Science, Aarhus University, Denmark.

¹Throughout this paper, we use $\log_x y$ to refer to the value $\max(1, \log_x y)$.

accesses required to answer a query using a disk-based data structure. This is the focus in this paper. In particular, we design and analyze our data structures in the *input/output model* (I/O model) of [1]. In this 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 *I/O operations* (I/Os) it performs. Throughout this paper, we use N to denote the input size, $n := N/B$ to denote the input size measured in blocks, and $m := M/B$ to denote the memory size measured in blocks. Aggarwal and Vitter [1] showed that comparison-based sorting of N elements in the I/O model takes $\text{sort}(N) = \Theta(n \log_m n)$ I/Os, while arranging N elements according to a given permutation takes $\text{perm}(N) = \Theta(\min(N, \text{sort}(N)))$ I/Os.

1.1 Previous Work. It seems that, even though top- K range reporting is a natural problem with practical applications, it has received little theoretical attention so far. The range minimum problem is a special case of top- K range reporting with $K = 1$. For this problem, a number of linear-space data structures with constant-time (and, hence, constant-I/O) queries have been obtained [5, 10, 14].

In [6], Brodal *et al.* studied ordered top- K range reporting in the word-RAM model and presented a linear-space data structure with query time $O(1 + K)$. In the pointer machine model, a priority search tree [12] combined with Frederickson's $O(K)$ -time algorithm for finding the K smallest elements in a binary heap-ordered tree [8] results in a linear-space data structure for unordered top- K range reporting with query bound $O(\log N + K)$. These K elements can then be sorted in $O(K \log K)$ time, so that the same data structure also supports ordered top- K range reporting queries in $O(\log N + K \log K)$ time. We are not aware of any non-trivial results on data structures for ordered top- K range reporting in the pointer machine or I/O model with an optimal dependence of the query bound on the output size.

1.2 New Results. In this paper, we present nearly matching space upper and lower bounds for ordered top- K range reporting data structures in the I/O model with a query bound of $O(\log_B N + fK/B)$ I/Os, for some parameter $1 \leq f \leq \log_m n$. In particular, we present an $O((N/f) \log_m n)$ -space data structure with this query bound in Section 3. For $f = 1$, this gives an $O(N \log_m n)$ -space data structure with a query bound

of $O(\log_B N + K/B)$ I/Os. For $f = \log_m n$, the data structure uses linear space, but at the expense of an increased query bound of $O(\log_B N + (K/B) \log_m n)$ I/Os. Our main contribution is to show that the space-query trade-off of our data structure is nearly optimal. In particular, we prove in Section 4 that any data structure with a query bound of $O(\log^\alpha n + fK/B)$ I/Os, for some constant α , has to use $\Omega\left(N \frac{f^{-1} \log_m n}{\log(f^{-1} \log_m n)}\right)$ space, as long as $B = \Omega(\log N)$. In general, this leaves a gap of $O(\log_m M \log \log_M n)$ to the upper bound. Under the common tall cache assumption ($M \geq B^{1+\epsilon}$, for some constant $\epsilon > 0$), the gap is $O(\log \log_m n)$. Our lower bound holds under the assumption of indivisibility of records; that is, to output an element, the query procedure must visit a cell in the data structure that stores this element. See Section 4 for more details.

As part of our upper bound construction, we present a linear-space data structure that achieves a query bound of $O(\log_B N + K/B)$ I/Os for *unordered* top- K range reporting; see Section 2. This demonstrates that the ordering requirement is exactly what makes ordered top- K range reporting hard. Our lower bound proof emphasizes this fact further, as it uses counting arguments similar to the ones used to prove the permutation lower bound in the I/O model [1]. The difference to that purely counting-based proof is that we cannot argue that an ordered range reporting data structure has to be able to report all $N!$ permutations of the elements in A . Indeed, only $O(N^3)$ different combinations of i , j , and K are possible, leading to only $O(N^3)$ different queries. Instead, we consider a hierarchy of query sizes and prove that, if the input is a random permutation of the integers between 1 and N , then (almost) independent data structures must be stored for the different query sizes. In other words, what needs to be stored to answer queries of one size efficiently is almost useless for answering queries of significantly larger or smaller size. For $\Omega\left(\frac{f^{-1} \log_m n}{\log(f^{-1} \log_m n)}\right)$ different query sizes, this gives the desired space lower bound.

2 Unordered Top- K Range Reporting

First we present a linear-space data structure that can answer unordered top- K range reporting queries using $O(\log_B N + K/B)$ I/Os. This data structure is used as part of our ordered top- K range reporting data structure described in Section 3.

THEOREM 2.1. *There exists a data structure that uses linear space to store a sequence A of N elements and is able to report the K smallest elements in the subsequence $A[i, j]$ using $O(\log_B N + K/B)$ I/Os, for any triple (i, j, K) with $1 \leq i \leq j \leq N$ and $1 \leq K \leq j - i + 1$.*

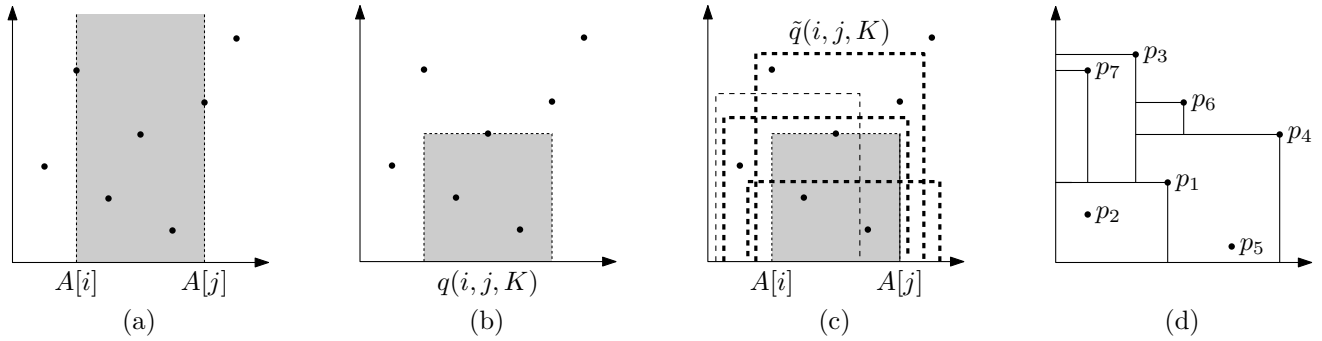


Figure 1: (a) The representation of the input array A . The range corresponding to the query array $A[i, j]$ is shaded in grey. (b) The three-sided query $q(i, j, K)$ is shaded in grey. (c) The dashed lines bound cells in the shallow cutting. Fat dashed lines bound the cells whose x -ranges cover $q(i, j, K)$. $\tilde{q}(i, j, K)$ is the cell with the highest top boundary among them. (d) An example of the subdivision \mathcal{A}_k .

We take the following view of the problem. We denote the i th element in A by a_i and consider the pair (i, a_i) to be a point in the plane; see Figure 1(a). Let $\mu_K(i, j)$ be the K th smallest element in $A[i, j]$. Then the K smallest elements in $A[i, j]$ are exactly the points in the three-sided query range $q(i, j, K) := [i, j] \times (-\infty, \mu_K(i, j)]$; see Figure 1(b). Given this query range, we could use an external priority search tree [4], which uses linear space and supports three-sided range reporting queries using $O(\log_B N + K/B)$ I/Os, to find the points in $q(i, j, K)$. Unfortunately, finding $\mu_K(i, j)$ does not seem to be any easier than finding the K smallest elements in $A[i, j]$.

To avoid this problem, we construct a data structure that consists of two parts. The first part is a data structure to find a three-sided query range $\tilde{q}(i, j, K)$ that covers $q(i, j, K)$ and contains $O(K)$ points; see Figure 1(c). Below, we describe such a data structure that uses linear space and can identify such a query range $\tilde{q}(i, j, K)$ using $O(\log_B N)$ I/Os, given only the triple (i, j, K) . The second part is an external priority search tree storing the input points. Given the query range $\tilde{q}(i, j, K)$, we can use the external priority search tree to retrieve the $O(K)$ points in $\tilde{q}(i, j, K)$ using $O(\log_B N + K/B)$ I/Os. Next, we can eliminate the points outside the x -range $[i, j]$ using a single scan of this point list. Then we apply linear-time selection [7] to the remaining points to find the K lowest points among them, which are exactly the points in $q(i, j, K)$. This takes $O(K/B)$ I/Os. Both parts of our data structure use linear space, and the total cost of the different parts of the query procedure is $O(\log_B N + K/B)$. Thus, to prove Theorem 2.1, it remains to describe the data structure that finds $\tilde{q}(i, j, K)$.

Our data structure for finding $\tilde{q}(i, j, K)$ is based on shallow cuttings [11]. In the context of three-sided range reporting, a *shallow K -cutting* is a collection of $O(N/K)$ three-sided ranges, called *cells*, such that each contains $O(K)$ points and, for every three-sided range q containing at most K points, there exists a cell that completely covers q . In the appendix, we describe a simple procedure for constructing a shallow cutting for a point set based on a construction from [2, 4]. Using standard I/O-efficient techniques, this construction can be implemented to take $O(N/K + \text{sort}(N))$ I/Os.

Let $\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_t$ be shallow cuttings, where $t := \lceil \log N \rceil$ and \mathcal{C}_h is a shallow 2^h -cutting. By the properties of shallow cuttings, there exists a cell in \mathcal{C}_k , where $k := \lceil \log K \rceil$, that covers $q(i, j, K)$ and contains $O(K)$ points. Thus, we can use such a cell in \mathcal{C}_k as the query range $\tilde{q}(i, j, K)$. What we need is a method to identify such a cell, given only the x -range $[i, j]$. Let $\mathcal{C}_{i,j,k}$ be the subset of cells $C \in \mathcal{C}_k$ whose x -ranges contain the x -range $[i, j]$; see Figure 1(c). We choose $\tilde{q}(i, j, K)$ to be the cell in $\mathcal{C}_{i,j,k}$ with maximum top boundary. This guarantees that $\tilde{q}(i, j, K)$ covers $q(i, j, K)$ because there exists a cell in \mathcal{C}_k that covers $q(i, j, K)$, only cells in $\mathcal{C}_{i,j,k}$ can cover $q(i, j, K)$, and, if the query with highest top boundary in $\mathcal{C}_{i,j,k}$ does not cover $q(i, j, K)$, then none of the cells in $\mathcal{C}_{i,j,k}$ does.

2.1 Finding $\tilde{q}(i, j, K)$. The x -range $[x_1, x_2]$ of a cell $C := [x_1, x_2] \times (-\infty, y]$ in \mathcal{C}_k contains the interval $[i, j]$ if and only if $x_1 \leq i$ and $j \leq x_2$. A standard transformation turns finding $\tilde{q}(i, j, K)$ into a 2-d dominance problem: map each cell $C \in \mathcal{C}_k$ to the point $p_C := (-x_1, x_2)$ and define the *weight* $w(p_C)$ of point p_C to be the top boundary y of C . Let P_k be the set of points obtained from \mathcal{C}_k in this manner. Then $\mathcal{C}_{i,j,k}$ is the set of cells

in \mathcal{C}_k whose corresponding points in P_k dominate the point $(-i, j)$, and $\tilde{q}(i, j, K)$ is the cell corresponding to the point with maximum weight among them. Thus, we need a data structure that solves this *max-dominance* problem.

2.2 Max-Dominance. Given the point set P_k , we define a planar subdivision \mathcal{A}_k as follows; see Figure 1(d). For every point $p \in P_k$, let $D(p)$ be the region dominated by p . Now let p_1, p_2, \dots be the sequence of points in P_k , sorted by decreasing weight. We associate a region $R(p_i) := D(p_i) \setminus \bigcup_{j=1}^{i-1} D(p_j)$ with every point $p_i \in P_k$. \mathcal{A}_k is the subdivision defined by this set of regions. The following observation is the basis for using the subdivision \mathcal{A}_k to find $\tilde{q}(i, j, K)$.

OBSERVATION 1. *A point $p_i \in P_k$ is the point with maximum weight among the points in P_k that dominate a query point q if and only if $q \in R(p_i)$. Furthermore, the complexity of the subdivision \mathcal{A}_k is $O(N/2^k)$.*

Proof. The first claim follows immediately from the definition of \mathcal{A}_k . To prove the second claim, we observe that \mathcal{A}_k is a subgraph of the subdivision obtained by inserting the points p_1, p_2, \dots by decreasing weight and, for each point shooting rays to the left and down until they hit an edge already in the subdivision. These contact points become vertices of the subdivision. Using this procedure, every point in P_k adds at most three vertices to the subdivision, and \mathcal{A}_k is a planar straight-line graph with $O(N/2^k)$ vertices. \square

By Observation 1, we can use a planar point location data structure on \mathcal{A}_k to identify $\tilde{q}(i, j, K)$. Data structures that use $O(|\mathcal{A}_k|) = O(N/2^k)$ space to represent \mathcal{A}_k and support point location queries using $O(\log_B N)$ I/Os are presented in [3, 9]. Thus, storing one such data structure for each of the arrangements $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_t$ corresponding to the shallow cuttings $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_t$ results in the desired linear-space data structure that can identify $\tilde{q}(i, j, K)$ using $O(\log_B N)$ I/Os.

3 Ordered Top- K Range Reporting

For ordered top- K range reporting, we prove the following result.

THEOREM 3.1. *There exists a data structure that uses $O((N/f) \log_m n)$ space to store a sequence A of N elements, for a parameter $1 \leq f \leq \log_m n$, and is able to report the K smallest elements in the subsequence $A[i, j]$ in sorted order using $O(\log_B N + fK/B)$ I/Os, for any triple (i, j, K) with $1 \leq i \leq j \leq N$ and $1 \leq K \leq j - i + 1$.*

For $K \leq Mm^f$, we can use the data structure for unordered top- K range reporting to answer ordered top- K range reporting queries. First we retrieve the K minimum elements in the query range using $O(\log_B N + K/B)$ I/Os; then we sort them, which takes $O((K/B) \log_m(K/M)) = O(fK/B)$ I/Os using standard external merge sort [1]. Thus, it remains to describe a data structure for $K > Mm^f$.

The first part of our data structure is the data structure we used in the previous section to identify the query range $\tilde{q}(i, j, K)$, given the triple (i, j, K) . Since $|\tilde{q}(i, j, K)| = O(K)$, it suffices to retrieve the points in $\tilde{q}(i, j, K)$ sorted by their y -coordinates; then we scan these points using $O(K/B)$ I/Os, discard all points not in the x -range $[i, j]$, and report the first K points among the remaining points. Next we describe an $O((N/f) \log_m n)$ -space data structure that can be used to retrieve the points in $\tilde{q}(i, j, K)$ in sorted order, using $O(fK/B)$ I/Os. This proves Theorem 3.1.

3.1 The Data Structure. Our data structure is based on the same set of shallow cuttings $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_t$ used in the unordered top- K range reporting data structure. Here we assume each shallow cutting \mathcal{C}_h has the following additional property: every three-sided query range q containing K points can be covered using $O(\lceil K/2^h \rceil)$ cells of \mathcal{C}_h . In the appendix, we describe a method based on a construction in [2, 4] to obtain this type of shallow cutting.

Now we consider a subset of these shallow cuttings, $\mathcal{C}_{\ell_1}, \dots, \mathcal{C}_{\ell_{t'}}$, where $\ell_i := \lfloor \log(Mm^{f^i}) \rfloor$ and, thus, $t' = O(f^{-1} \log_m n)$. Each shallow cutting \mathcal{C}_{ℓ_i} is a shallow $\Theta(Mm^{f^i})$ -cutting. For each such shallow cutting \mathcal{C}_{ℓ_i} and each cell $C \in \mathcal{C}_{\ell_i}$, we store the points in C in y -sorted order. This takes $O(N)$ space for one shallow cutting \mathcal{C}_{ℓ_i} and, thus, $O(Nt') = O((N/f) \log_m n)$ space in total. For every shallow cutting \mathcal{C}_h with $\log(Mm^f) \leq h \leq t$, let j be the index such that $\ell_j \leq h < \ell_{j+1}$. Every cell $C \in \mathcal{C}_h$ can be covered using $O(2^h/2^{\ell_j}) = O(m^f)$ cells in \mathcal{C}_{ℓ_j} , and we store pointers to these cells with the cell C in \mathcal{C}_h . Each cell in \mathcal{C}_h stores $O(m^f)$ pointers, and the total number of cells in shallow cuttings \mathcal{C}_h with $\log(Mm^f) \leq h \leq t$ is $O(N/(Mm^f))$. Hence, these pointers use $O(N/M)$ space in total, and the size of the data structure is dominated by the size of the sorted point lists for the shallow cuttings $\mathcal{C}_{\ell_1}, \mathcal{C}_{\ell_2}, \dots, \mathcal{C}_{\ell_{t'}}$, which is $O((N/f) \log_m n)$.

3.2 The Query Procedure. To retrieve the point list of a shallow cutting cell $\tilde{q}(i, j, K) \in \mathcal{C}_k$ using $O(f2^k/B) = O(fK/B)$ I/Os, let j be the index such that $\ell_j \leq k < \ell_{j+1}$. We follow the pointers from $\tilde{q}(i, j, K)$ to the $O(2^k/2^{\ell_j})$ cells in \mathcal{C}_{ℓ_j} that cover

$\tilde{q}(i, j, K)$ and merge their point lists using standard m -way merging [1]. Then we scan the resulting sorted point list and discard duplicates and points not in $\tilde{q}(i, j, K)$. Since $O(2^k/2^{\ell_j}) = O(m^f)$, the merging of these $O(m^f)$ lists requires $O(m^f + (K'/B) \log_m(m^f)) = O(m^f + fK'/B)$ I/Os, where K' is the number of points in the merged lists. The $O(m^f)$ term accounts for the random accesses required to retrieve the first block of each list to be merged. Since we merge $O(2^k/2^{\ell_j})$ point lists of cells in \mathcal{C}_{ℓ_j} and every cell of \mathcal{C}_{ℓ_j} contains $O(2^{\ell_j})$ points, we have $K' = O(2^k)$, that is, the merging cost is $O(m^f + f2^k/B) = O(m^f + fK/B)$. Now it suffices to observe that $m^f = O(K/B)$ because $K \geq m^f M$ and $M \geq B$. This proves that the cost of retrieving the sorted point list of $\tilde{q}(i, j, K)$ is $O(fK/B)$.

4 A Lower Bound for Ordered Range Reporting

In this section, we prove a lower bound on the size of any data structure that achieves a query bound of $O(\log^\alpha n + fK/B)$ I/Os for ordered top- K range reporting, where α is a constant and $1 \leq f \leq \log_m n$. The lower bound matches the upper bound achieved in Theorem 3.1 up to a factor of $O(\log_m M \log \log_M n)$, which is $O(\log \log_m n)$ when $M \geq B^{1+\varepsilon}$, for a constant $\varepsilon > 0$ (the tall cache assumption).

Ordered range reporting is a special case of ordered top- K range reporting that simply asks to report *all* elements in the query range in sorted order. We prove our lower bound for any ordered range reporting data structure with the above query bound, which implies the same lower bound for any ordered top- K range reporting data structure.

THEOREM 4.1. *Any data structure capable of answering ordered range reporting queries over a sequence of N elements using $O(\log^\alpha n + fK/B)$ I/Os, for a constant α , $1 \leq f \leq \log_m n$, and $B = \Omega(\log N)$, must use $\Omega\left(N \frac{f^{-1} \log_M n}{\log(f^{-1} \log_M n)}\right)$ space in the worst case.*

4.1 Lower Bound Model. Our lower bound holds for any data structure in the I/O model with the additional assumption of *indivisibility of records*. This means that we assume the data structure stores the data elements in a linear sequence of *cells* (disk locations), each cell stores at most one element, and the query procedure is only allowed to move or copy data elements but not create any data elements without accessing them (e.g., using arithmetic operations). Apart from this restriction on storing data elements, the data structure may store any kind of book-keeping information, and we place no restriction on the operations used to manipulate this information.

We can view a sequence of I/O operations performed by an algorithm \mathcal{A} as a transformation of the sequence of occupied disk blocks and, hence, of the sequence σ of elements stored in these blocks into a new sequence σ' . We say that \mathcal{A} *generates* a sequence ρ from a subsequence τ of σ if ρ is a (not necessarily contiguous) subsequence of σ' and each element of ρ can be traced back as being copied from an element of τ . (By the indivisibility assumption, every element of σ' can be traced back to a unique element of σ .)

4.2 Proof of Theorem 4.1. Our lower bound is in fact a permutation lower bound, showing that it is impossible to build a small data structure capable of quickly permuting each subsequence of the input sequence into its sorted order. A lower bound on the number of I/Os necessary to transform an input permutation into a target permutation was shown in [1]. While our approach is quite different, we first describe the intuition behind that proof, as it serves as a starting point for our argument.

For a fixed permutation stored on disk, it is not difficult to see that only a bounded number of permutations can be generated from it using a single I/O. In other words, we can explore only a small fraction of the space of all permutations using a single I/O from a given start permutation. Since the permutation space contains $N!$ different permutations, this observation can be used to show a non-trivial lower bound on the number of I/Os required to generate any permutation from a fixed starting permutation.

In the case of ordered range reporting, we have only $O(N^2)$ different query ranges, that is, the data structure only needs to be able to report $O(N^2)$ different output sequences quickly, which makes the above counting argument ineffective. To overcome this difficulty, we consider a fixed random permutation π and show that, with non-zero probability, any efficient ordered range reporting data structure storing π must use the space stated in Theorem 4.1. To prove this, we derive $h = \Omega\left(\frac{f^{-1} \log_M n}{\log(f^{-1} \log_M n)}\right)$ permutations $\sigma_1(\pi), \sigma_2(\pi), \dots, \sigma_h(\pi)$ of the elements in π and show that any data structure that can answer ordered range reporting queries over π using $O(\log^\alpha n + fK/B)$ I/Os, for a constant α , must be able to generate each permutation $\sigma_i(\pi)$, for $1 \leq i \leq h$, using $O(fn)$ I/Os. For a random permutation π , these permutations $\sigma_1(\pi), \sigma_2(\pi), \dots, \sigma_h(\pi)$ will be “far apart” in the permutation space, which makes it impossible to store only one permutation in the data structure that is close even to two of these permutations (i.e., can be used as a starting point to generate both permutations quickly). This suggests that the best strategy is to store one permutation close to each

permutation $\sigma_i(\pi)$ in the data structure, which requires $\Omega(Nh) = \Omega\left(N \frac{f^{-1} \log_M n}{\log(f^{-1} \log_M n)}\right)$ space. Of course, the data structure might be able to store a short sequence of elements (with multiple copies of each element) such that, for each i , a permutation close to $\sigma_i(\pi)$ can be found as a subsequence of this sequence. The main difficulty is to prove that any sequence containing such permutations close to $\sigma_1(\pi), \sigma_2(\pi), \dots, \sigma_h(\pi)$ as subsequences must have length $\Omega(Nh)$, that is, is not significantly shorter than the naive concatenation of $\sigma_1(\pi), \sigma_2(\pi), \dots, \sigma_h(\pi)$.

We define these sequences $\sigma_1(\pi), \sigma_2(\pi), \dots, \sigma_h(\pi)$ as follows. Let $N > N_1 \geq N_2 \geq \dots \geq N_h \geq 1$ be parameters to be chosen later and, for simplicity, assume N_i divides N . For each $1 \leq i \leq h$, we divide π into N/N_i contiguous subsequences of length N_i , called *level- i pieces*. The permutation $\sigma_i(\pi)$, called a *level- i permutation*, is obtained from π by sorting its level- i pieces.

Now consider a fixed data structure $\mathcal{D}(\pi)$ constructed over π (which is simply a sequence of elements stored on disk) and an algorithm that generates the sequence $\sigma_i(\pi)$ from $\mathcal{D}(\pi)$. Every element in $\sigma_i(\pi)$ can be traced back to an *origin cell* in $\mathcal{D}(\pi)$. We use $\mathcal{D}_i(\pi)$ to denote the set of these origin cells for all elements of $\sigma_i(\pi)$. Using our terminology, this means that the algorithm generates $\sigma_i(\pi)$ from $\mathcal{D}_i(\pi)$. We show that, for a random permutation π and a fixed level i , the probability that $|\mathcal{D}_i(\pi) \cap (\mathcal{D}_1(\pi) \cup \mathcal{D}_2(\pi) \cup \dots \cup \mathcal{D}_{i-1}(\pi))| \leq N/2$ is high, for any data structure $\mathcal{D}(\pi)$ that can generate each of the permutations $\sigma_1(\pi), \sigma_2(\pi), \dots, \sigma_i(\pi)$ using $O(fn)$ I/Os. This implies that there exists a permutation π such that this condition holds for all $1 < i \leq h$. Thus, the set of cells $\mathcal{D}_1(\pi) \cup \mathcal{D}_2(\pi) \cup \dots \cup \mathcal{D}_h(\pi)$ has size at least $hN/2$, that is, the size of the data structure is at least $hN/2 = \Omega\left(N \frac{f^{-1} \log_M n}{\log(f^{-1} \log_M n)}\right)$. Since this is true for every data structure that can generate $\sigma_1(\pi), \sigma_2(\pi), \dots, \sigma_h(\pi)$ efficiently, Theorem 4.1 follows.

To bound the probability that $|\mathcal{D}_i(\pi) \cap (\mathcal{D}_1(\pi) \cup \mathcal{D}_2(\pi) \cup \dots \cup \mathcal{D}_{i-1}(\pi))| > N/2$, we fix a uniform random permutation π , a data structure $\mathcal{D}(\pi)$, and two levels $1 \leq k < i \leq h$, and bound $|\mathcal{D}_i(\pi) \cap \mathcal{D}_k(\pi)|$. Let $\sigma_k^\circ(\pi)$ be the subsequence of $\sigma_k(\pi)$ whose origin cells belong to $\mathcal{D}_i(\pi) \cap \mathcal{D}_k(\pi)$. Since $\sigma_k(\pi)$ can be generated from $\mathcal{D}_k(\pi)$ using $O(fn)$ I/Os, $\sigma_k^\circ(\pi)$ can also be generated from $\mathcal{D}_i(\pi) \cap \mathcal{D}_k(\pi)$ using $O(fn)$ I/Os. Since $\sigma_i(\pi)$ can be generated from $\mathcal{D}_i(\pi)$ using $O(fn)$ I/Os, this implies that $\sigma_k^\circ(\pi)$ can be generated from $\sigma_i(\pi)$ using $O(fn)$ I/Os: first we invert the I/O sequence generating $\sigma_i(\pi)$ from $\mathcal{D}_i(\pi)$ to obtain $\mathcal{D}_i(\pi)$ from $\sigma_i(\pi)$, and then we apply the I/O sequence that generates $\sigma_k^\circ(\pi)$ from $\mathcal{D}_i(\pi) \cap \mathcal{D}_k(\pi)$. Now, for $fn = o(\text{perm}(N))$,

it is impossible to generate every permutation of N elements from $\sigma_i(\pi)$ using $O(fn)$ I/Os. We prove that, with high probability, $\sigma_k(\pi)$ is “sufficiently different” from $\sigma_i(\pi)$ that we also cannot generate $\sigma_k^\circ(\pi)$ from $\sigma_i(\pi)$ using $O(fn)$ I/Os unless $\sigma_k^\circ(\pi)$ contains only a small portion of the elements of $\sigma_k(\pi)$, that is, unless $|\mathcal{D}_i(\pi) \cap \mathcal{D}_k(\pi)|$ is small. By applying this argument to all pairs (i, k) with $1 \leq k < i$, we obtain the desired bound on the size of the intersection between $\mathcal{D}_i(\pi)$ and $\mathcal{D}_1(\pi) \cup \mathcal{D}_2(\pi) \cup \dots \cup \mathcal{D}_{i-1}(\pi)$.

In the remainder of this section, we define the parameters N_1, N_2, \dots, N_h used to construct the permutations $\sigma_1(\pi), \sigma_2(\pi), \dots, \sigma_h(\pi)$, introduce some more notation, state our main lemma (Lemma 4.2), and prove that it implies Theorem 4.1. In Section 4.3, we prove Lemma 4.2.

Let C be a constant, $\gamma := Cf \log(f^{-1} \log_M n)$, $t := M^\gamma$, and $h := \lfloor \log_t(n / \log^\alpha n) \rfloor$. For $1 \leq i \leq h$, we define $N_i := N/t^i$. Then, for $1 \leq i \leq h$ and $0 \leq j < t^i$, we use $\pi_{i,j}$ to denote the j th level- i piece $\langle \pi(jN_i + 1), \pi(jN_i + 2), \dots, \pi((j+1)N_i) \rangle$ of π , and $\sigma_{i,j}(\pi)$ to denote the sequence obtained by sorting the elements in $\pi_{i,j}$. Thus, $\sigma_i(\pi)$ is the concatenation of the sequences $\sigma_{i,0}(\pi), \sigma_{i,1}(\pi), \dots, \sigma_{i,t^i-1}(\pi)$. We also refer to $\sigma_{i,j}(\pi)$ as a level- i piece of $\sigma_i(\pi)$.

LEMMA 4.1. *Any data structure that supports ordered range reporting queries over π using $O(\log^\alpha n + fK/B)$ I/Os can generate each of the sequences $\sigma_1(\pi), \sigma_2(\pi), \dots, \sigma_h(\pi)$ using $O(fn)$ I/Os.*

Proof. Consider a sequence $\sigma_i(\pi)$. We can generate $\sigma_i(\pi)$ by reporting each sequence $\sigma_{i,j}(\pi)$ in turn, for $0 \leq j < t^i$. In particular, for $0 \leq j < t^i$, we report $\sigma_{i,j}(\pi)$ and then copy $\sigma_{i,j}(\pi)$ to a new sequence of $O(N_i/B)$ blocks so that the blocks containing the sequence $\sigma_{i,j}(\pi)$ succeed the blocks containing the sequence $\sigma_{i,j'}(\pi)$, for all $j' < j$. The resulting sequence of $O(t^i N_i/B) = O(N/B)$ blocks then stores the sequence $\sigma_i(\pi)$. Since $\sigma_{i,j}(\pi)$ is the result of an ordered range reporting query over π with query interval $[jN_i + 1, (j+1)N_i]$, each subsequence $\sigma_{i,j}(\pi)$ can be reported using $O(\log^\alpha n + fN_i/B)$ I/Os. Thus, generating $\sigma_i(\pi)$ in this manner takes $O(t^i(\log^\alpha n + fN_i/B)) = O(t^i \log^\alpha n + fn) = O(fn)$ I/Os. \square

By Lemma 4.1, any lower bound on the size of a data structure that can generate each of the permutations $\sigma_1(\pi), \sigma_2(\pi), \dots, \sigma_h(\pi)$ using $O(fn)$ I/Os is also a lower bound on the size of any ordered range reporting data structure over π with a query bound of $O(\log^\alpha n + fK/B)$ I/Os. The following is our main lemma, which we prove in Section 4.3. Here we prove that this lemma implies Theorem 4.1.

LEMMA 4.2. For a random permutation π , any two indices $1 \leq k < i \leq h$, and any data structure $\mathcal{D}(\pi)$ that can generate $\sigma_i(\pi)$ and $\sigma_k(\pi)$ using $O(fn)$ I/Os, the number of cells in $\mathcal{D}_i(\pi) \cap \mathcal{D}_k(\pi)$ is at most $\frac{N}{3(i-k)\log h}$ with probability at least $1 - 1/N$, assuming $B = \Omega(\log N)$.

Lemma 4.2 implies that, for a uniform random permutation π , the number of cells shared between $\mathcal{D}_i(\pi)$ and $\mathcal{D}_1(\pi) \cup \mathcal{D}_2(\pi) \cup \dots \cup \mathcal{D}_{i-1}(\pi)$ is at most

$$\sum_{k=1}^{i-1} \frac{N}{3(i-k)\log h} \leq \left(\frac{N}{3\log h} \right) (\ln i + 1) \leq N/2$$

with probability at least $1 - (i-1)/N$. Thus, the probability that $|\mathcal{D}_i(\pi) \cap (\mathcal{D}_1(\pi) \cup \mathcal{D}_2(\pi) \cup \dots \cup \mathcal{D}_{i-1}(\pi))| \leq N/2$, for all $1 \leq i \leq h$, is at least $1 - h^2/N > 0$, that is, there exists a permutation π so that this is true. As we argued previously, this implies that the size of the data structure $\mathcal{D}(\pi)$ is at least

$$\frac{hN}{2} = \Omega\left(N \frac{f^{-1} \log_M n}{\log(f^{-1} \log_M n)}\right)$$

for this permutation. This proves Theorem 4.1.

4.3 Proof of Lemma 4.2. To prove Lemma 4.2, we fix two levels $1 \leq k < i \leq h$. We begin by proving that, for a uniform random permutation π , the permutation $\sigma_i(\pi)$ is a uniform random level- i permutation, that is, $\sigma_i(\pi)$ is drawn uniformly at random from the set of all possible level- i permutations (permutations composed of sorted level- i pieces). Once $\sigma_i(\pi)$ is fixed, so is $\sigma_k(\pi)$ because $\sigma_k(\pi)$ is obtained from $\sigma_i(\pi)$ by sorting its level- k pieces. The remainder of the proof then shows that, for a uniform random level- i permutation σ_i , its corresponding level- k permutation σ_k is sufficiently different from σ_i that $\mathcal{D}_i \cap \mathcal{D}_k$ must be small for any data structure \mathcal{D} that can generate both σ_i and σ_k using $O(fn)$ I/Os.

LEMMA 4.3. Consider a level- i permutation σ'_i and a uniform random permutation π . With probability $(N_i!)^{t^i}/N!$, $\sigma_i(\pi) = \sigma'_i$. The number of distinct level- i permutations is $t^{iN}/O(2^N)$.

Proof. To prove the first part of the lemma, observe that $\sigma_i(\pi)$ is independent of the order of the elements in the level- i pieces of π because $\sigma_i(\pi)$ is obtained from π by sorting the elements in these pieces. What matters is the set of elements that occur in each level- i piece of π . Thus, there are $(N_i!)^{t^i}$ different permutations π that define the same level- i permutation $\sigma_i(\pi)$, and the probability that $\sigma_i(\pi) = \sigma'_i$, for a fixed level- i

permutation σ'_i and a uniform random permutation π , is $(N_i!)^{t^i}/N!$.

The second part of the lemma follows because the number of level- i permutations is $N!/(N_i!)^{t^i}$. Using Stirling's approximation ($N! = (N/e)^N \cdot \Theta(\sqrt{N})$), this gives a bound of

$$\frac{(N/e)^N \Theta(\sqrt{N})}{(N_i/e)^{N_i t^i} \Theta(\sqrt{N_i})^{t^i}} \geq t^{iN}/2^{O(N)}$$

because $N_i \cdot t^i = N$ and $\Theta(\sqrt{N_i})^{t^i} = 2^{O(N)}$. \square

By Lemma 4.3, we can ignore the random permutation π used to define $\sigma_i(\pi)$ and $\sigma_k(\pi)$ and instead reason about a uniform random level- i permutation σ_i , its corresponding level- k permutation σ_k , and a data structure \mathcal{D} that can generate both σ_i and σ_k using $O(fn)$ I/Os. Once again, let \mathcal{D}_i and \mathcal{D}_k denote the sets of cells in \mathcal{D} that are used to generate σ_i and σ_k , respectively, and let σ_i° and σ_k° be the subsequences of σ_i and σ_k that are generated from the cells in $\mathcal{D}_i \cap \mathcal{D}_k$. To distinguish the elements of σ_i° and σ_k° from the remaining elements in σ_i and σ_k , we mark the elements in σ_i° and σ_k° and leave the remaining elements unmarked.

As argued before, the fact that σ_i can be generated from \mathcal{D}_i using $O(fn)$ I/Os and σ_k can be generated from \mathcal{D}_k using $O(fn)$ I/Os implies that σ_k° can be generated from σ_i using $O(fn)$ I/Os. W.l.o.g., we can assume that the I/O sequence generating σ_k° from σ_i does not overwrite disk blocks. If it does, we can alter it to instead create new blocks right next to the blocks it would otherwise have overwritten. It is easy to see that this altered I/O sequence still generates σ_k° . The advantage of this view is that we can consider not only the effect this I/O sequence has on σ_i° , which is to generate σ_k° from it, but the effect it has on the entire sequence σ_i , which is to generate a new permutation of the elements in σ_i that arranges the marked elements in the same order as in σ_k .

To prove that $|\mathcal{D}_i \cap \mathcal{D}_k|$ cannot be too large, we consider a set of variables x_1, x_2, \dots, x_N . Each level- i permutation σ_i defines a particular assignment of values to these variables, which assigns the j th element in σ_i to the variable x_j . We mark a variable x_j if it is assigned a marked element of σ_i . Now, using $O(fn)$ I/Os, we can generate a subset of all permutations of the variables x_1, x_2, \dots, x_N . Each of these has 2^N corresponding *marked permutations*, each of which is obtained by marking a particular subset of the variables x_1, x_2, \dots, x_N in the permutation. We use \mathcal{P} to denote the set of all marked permutations generated in this way.

These marked permutations capture the following intuition. The permutation σ_i defines a particular as-

signment of elements to the variables x_1, x_2, \dots, x_N . Since we can generate σ_k° from σ_i using $O(fn)$ I/Os, there exists a permutation $\rho \in \mathcal{P}$ so that the permutation $\rho \circ \sigma_i$ obtained by assigning the j th element in σ_i to the variable x_j in ρ arranges the elements in σ_i° in the same order as in σ_k° . The marking of variables reflects which variables receive elements of σ_i° and, thus, need to be arranged in an order matching σ_k . Now, for a given marked permutation $\rho \in \mathcal{P}$ and a level- i permutation σ_i , $\rho \circ \sigma_i$ may or may not arrange the elements assigned by σ_i to variables marked by ρ in the order they appear in the level- k permutation σ_k corresponding to σ_i , that is, ρ may or may not generate σ_k° from σ_i . If it does, we call σ_i ρ -consistent; otherwise we don't.

The remainder of the proof of Lemma 4.2 consists of two steps. First we show that, for a given marked permutation $\rho \in \mathcal{P}$ with many marked variables, the number of ρ -consistent level- i permutations is small. Next, we show that the number of marked permutations is also small. By Lemma 4.3, the number of level- i permutations is large. Together, these three facts imply that, with high probability, a random level- i permutation σ_i is not ρ -consistent with respect to any marked permutation $\rho \in \mathcal{P}$ that has many marked variables, that is, $|\mathcal{D}_i \cap \mathcal{D}_k|$ must be small, for any data structure \mathcal{D} that can generate σ_i and its corresponding level- k permutation σ_k using $O(fn)$ I/Os.

LEMMA 4.4. *For a fixed marked permutation $\rho \in \mathcal{P}$ that marks βN variables, for some $0 \leq \beta \leq 1$, the number of level- i permutations that are ρ -consistent is $t^{k\beta N} t^{i(N-\beta N)} 2^{O(N)}$.*

Proof. For $0 \leq j < t^k$, let M_j be the set of variables among $x_{jN_k+1}, x_{jN_k+2}, \dots, x_{(j+1)N_k}$ that are marked by ρ , and let $m_j := |M_j|$. The elements assigned to $x_{jN_k+1}, x_{jN_k+2}, \dots, x_{(j+1)N_k}$ by a level- i permutation σ_i are exactly the elements of a level- k piece $\sigma_{k,j}$ of σ_k and are arranged in sorted order in $\sigma_{k,j}$. The requirement that the level- i permutation be ρ -consistent implies that, once the set of elements assigned by σ_i to the variables in M_j is fixed, there is only one way to assign these elements to these variables. Every level- k piece of σ_k is divided into t^{i-k} level- i pieces, and the elements in each such level- i piece occur in sorted order in σ_i . Therefore, once the set of elements assigned by σ_i to the unmarked variables in such a level- i piece is chosen, there is once again only one way to assign them. This leads to the following method of bounding the number of assignments to variables x_1, x_2, \dots, x_N defined by ρ -consistent level- i permutations.

For $0 \leq j' < t^{i-k}$, let $m_{j,j'}$ be the number of marked variables among $x_{jN_k+j'N_i+1}, x_{jN_k+j'N_i+2}, \dots, x_{jN_k+(j'+1)N_i}$. We have $m_j = \sum_{j'=0}^{t^{i-k}-1} m_{j,j'}$. For

each level- k piece, we construct an assignment to its variables by first choosing m_j elements to be assigned to the variables in M_j and then choosing the set of elements to be assigned to the unmarked variables in each level- i piece contained in this level- k piece. While it is not hard to see that not every assignment produced in this way corresponds to a level- i permutation (because we do not enforce any ordering constraints on the marked elements with respect to the unmarked elements in σ_i), the argument in the previous paragraph implies that every assignment representing a ρ -consistent level- i permutation can be generated in this way, giving us an upper bound on the number of ρ -consistent level- i permutations.

We begin by counting the number of possible assignments to the variables in the j th level- k piece, assuming we can choose the elements from a universe of size U . We denote this number by $a(U, j)$. We have

$$a(U, j) = \binom{U}{m_j} \cdot \binom{U - m_j}{N_i - m_{j,0}} \cdot \binom{U - m_j - (N_i - m_{j,0})}{N_i - m_{j-1}} \cdots \binom{U - m_j - (N_i - m_{j,0}) - \cdots - (N_i - m_{j,t^{i-j}-2})}{N_i - m_{j,t^{i-j}-1}},$$

where the first term accounts for the different assignments of values to marked variables and each of the t^{i-k} subsequent terms accounts for the different assignments to unmarked variables in one of the level- i pieces. The total number of assignments we can generate using this approach, counting all combinations of assignments to the t^k level- k pieces, is therefore

$$a(N) := a(N, 0)a(N - N_k, 1) \cdots a(N_k, t^k).$$

By expanding this expression and replacing each binomial coefficient $\binom{a}{b}$ with $\frac{a!}{b!(a-b)!}$, we obtain

$$(4.1) \quad a(N) = \frac{N!}{\prod_{j=0}^{t^k-1} \left(m_j! \cdot \prod_{j'=0}^{t^{i-k}-1} (N_i - m_{j,j'})! \right)}.$$

It can be verified that (4.1) is maximized when each $m_{j,j'}$ is roughly equal to m_j/t^{i-k} and each m_j is roughly equal to $\beta N/t^k = \beta N_k$. Note that this means that each $m_{j,j'}$ is roughly equal to $m_j/t^{i-k} = \beta N_i$. With these values and using Sterling's formula, we obtain an upper bound on $a(N)$ of

$$\frac{N^N}{N_k^{\beta N} N_i^{N-\beta N}} \cdot 2^{O(N)} = t^{k\beta N} t^{i(N-\beta N)} 2^{O(N)}. \quad \square$$

The next lemma bounds the number of marked permutations in \mathcal{P} .

LEMMA 4.5. Assuming $B = \Omega(\log N)$, the number of marked permutations of the variables x_1, x_2, \dots, x_N that can be generated using $O(fn)$ I/Os is $M^{O(fn)}$.

Proof. It suffices to prove that a single I/O increases the number of permutations that can be generated from the sequence $\langle x_1, x_2, \dots, x_N \rangle$ by a factor of $M^{O(B)}$. The number of permutations generated by $O(fn)$ I/Os is then $M^{O(B) \cdot O(fn)} = M^{O(fn)}$, and for each such permutation, there are 2^N ways of marking its elements. Thus, the number of marked permutations that can be generated from $\langle x_1, x_2, \dots, x_N \rangle$ using $O(fn)$ I/Os is $2^N M^{O(fn)} = M^{O(fn)}$.

Consider a given I/O operation, and assume there are T occupied disk blocks before this operation. A read operation only loads elements into memory and, thus, does not change the number of permutations of variables x_1, x_2, \dots, x_N found in the current set of disk blocks. A write operation chooses B of the M elements in memory to write to a new disk block, arranges them in one of $B!$ ways inside the disk block, and places the new disk block in one of $T + 1$ locations relative to the existing T blocks. (Remember we assumed we do not overwrite existing blocks.) This gives $(T + 1) \binom{M}{B} B! \leq (T + 1) M^B$ possibilities. For each such choice and each permutation π of x_1, x_2, \dots, x_N already present as a subsequence of cells in the T blocks before this write operation, we can choose a subset S of 2^B elements from the block just written. These elements are arranged as a particular sequence σ in this block. We can then delete the elements in S from π and insert the sequence σ in at most $N - |S| \leq N$ different positions with respect to the elements we did not delete from π . This gives another increase of the number of permutations represented by the current set of disk blocks by a factor of at most $N \cdot 2^B$, that is, a single I/O increases the number of permutations by a factor of at most $N(T + 1)(2M)^B$.

Since the variables x_1, x_2, \dots, x_N are initially stored in at most N disk blocks and $fn < N^2$, we have $O(N^2)$ blocks containing copies of the variables x_1, x_2, \dots, x_N at any point during a sequence of $O(fn)$ I/Os applied to the sequence $\langle x_1, x_2, \dots, x_N \rangle$. Therefore, one I/O can increase the number of permutations by a factor of at most $O(N^3)(2M)^B = M^{O(B)}$, since $B = \Omega(\log N)$. \square

By Lemma 4.5, there are $M^{O(fn)}$ different marked permutations in \mathcal{P} , and by Lemma 4.4, only $t^{k\beta N} t^{i(N-\beta N)} 2^{O(N)}$ level- i permutations are ρ -consistent, for each $\rho \in \mathcal{P}$ that marks βN variables. Thus, the total number of level- i permutations that are ρ -consistent for at least one $\rho \in \mathcal{P}$ that marks βN elements is $M^{O(fn)} t^{k\beta N} t^{i(N-\beta N)} 2^{O(N)} = M^{O(fn)} t^{k\beta N} t^{i(N-\beta N)}$. The term $t^{k\beta N} t^{i(N-\beta N)}$ decreases by a factor of $t^{i-k} \geq t$ if we increase βN by

one. Therefore, the number of level- i permutations that are ρ -consistent for at least one $\rho \in \mathcal{P}$ that marks at least βN variables is $O(M^{O(fn)} t^{k\beta N} t^{i(N-\beta N)}) = M^{O(fn)} t^{k\beta N} t^{i(N-\beta N)}$.

By Lemma 4.3, on the other hand, there are $t^{iN}/2^{O(N)}$ different level- i permutations. Thus, the probability that a uniform random level- i permutation is ρ -consistent, for some marked permutation $\rho \in \mathcal{P}$ that marks at least βN elements, is at most

$$\frac{M^{O(fn)} t^{k\beta N} t^{i(N-\beta N)}}{t^{iN}/2^{O(N)}} = \frac{M^{O(fn)}}{t^{(i-k)\beta N}}.$$

For $\beta := 1/(3(i-k)\log h)$, this equals $M^{O(fn)}/t^{N/(3\log h)}$, which is bounded by $1/N$ if we choose the constant C in the definition of t large enough. Thus, with probability at least $1 - 1/N$, a uniform random level- i permutation σ_i is ρ -consistent only for marked permutations $\rho \in \mathcal{P}$ that mark less than $N/(3(i-k)\log h)$ variables. This implies that, for a uniform random level- i permutation σ_i and any data structure \mathcal{D} that can generate σ_i and σ_k using $O(fn)$ I/Os, we have $|\mathcal{D}_i \cap \mathcal{D}_k| \leq N/(3(i-k)\log h)$ with probability $1 - 1/N$. This finishes the proof of Lemma 4.2 and, hence, of Theorem 4.1.

References

- [1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.
- [2] L. Arge, G. S. Brodal, R. Fagerberg, and M. Laustsen. Cache-oblivious planar orthogonal range searching and counting. In *Proceedings of the 21st ACM Symposium on Computational Geometry*, pages 160–169, 2005.
- [3] L. Arge, A. Danner, and S.-M. Teh. I/O-efficient point location using persistent B-trees. *ACM Journal of Experimental Algorithmics*, 8, 2003.
- [4] 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.
- [5] M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer-Verlag, 2000.
- [6] G. S. Brodal, R. Fagerberg, M. Greve, and A. López-Ortiz. Online sorted range reporting. In *Proceedings of the 20th International Symposium on Algorithms and Computation*, volume 5878 of *Lecture Notes in Computer Science*, pages 173–182. Springer-Verlag, 2009.
- [7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001.

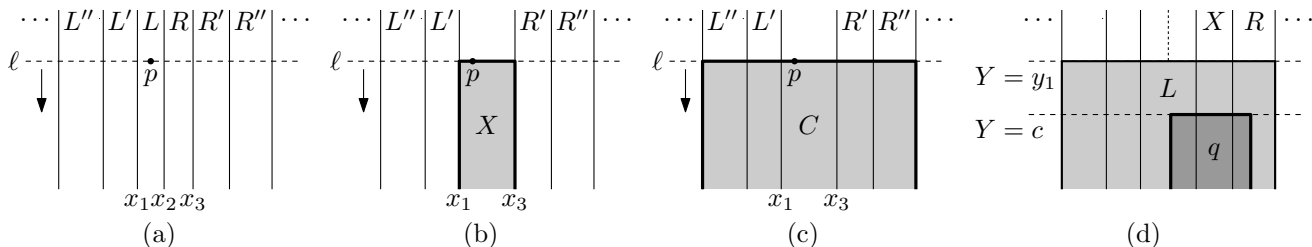


Figure 2: (a) The invariant is violated at point p . (b) The new active bucket X that replaces L and R . (c) The new shallow cutting cell C created together with bucket X . (d) The query q contains at most K points and intersects at most three active buckets L , X , and R . The line $Y = c$ denotes the position of the sweep line when it reaches the top boundary of q . The line $Y = y_1$ denotes the position of the sweep line when L was created. The region in the shallow cutting cell created with L is shaded in grey. If L is the most recently created active bucket among L , X , and R , then X and R were active at the time of its creation; thus, the shallow cutting cell created with L covers q .

[8] G. N. Frederickson. An optimal algorithm for selection in a min-heap. *Information and Computation*, 104(2):197–214, 1993.

[9] M. T. Goodrich, J.-J. T. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, pages 714–723, 1993.

[10] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

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

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

[13] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, 1999.

[14] B. Schieber and U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM Journal on Computing*, 17:1253–1262, 1988.

A Constructing Shallow K -Cuttings

Here we discuss how to obtain shallow cuttings with the properties required in Section 3 using a construction similar to one used in [2, 4]. For a parameter K , we want to construct a set \mathcal{C} of $O(N/K)$ three-sided ranges, called cells, that have the following properties: each cell $C \in \mathcal{C}$ contains $O(K)$ points and, for every three-sided range q that contains T points, there exists *one* cell in \mathcal{C} that contains q if $T \leq K$, and $O(T/K)$ cells in \mathcal{C} whose union covers q if $T > K$. W.l.o.g., we assume three-sided ranges are of the form $[x_1, x_2] \times (-\infty, y]$, that is, are open to the bottom.

We construct \mathcal{C} by sweeping a horizontal line ℓ across the plane. The sweep starts at $Y = +\infty$ and moves down towards $Y = -\infty$. Before the sweep, we

divide the plane into N/K vertical slabs containing K points each. For each such slab covering the x -range $[x_1, x_2]$, we create a *bucket*, which is a three-sided range $[x_1, x_2] \times (-\infty, +\infty)$. These buckets can be active or inactive; initially, all buckets are active. We also create a cell of \mathcal{C} for each group of five consecutive buckets. This cell is the union of these buckets. During the sweep, we call a point *active* if it is below the sweep line. We maintain the invariant that the number of active points in two adjacent active buckets is more than K .

Now consider the event when two adjacent buckets $L = [x_1, x_2] \times (-\infty, y_1]$ and $R = [x_2, x_3] \times (-\infty, y_2]$ start to violate this invariant, that is, $L \cup R$ contains only K active points. This happens when the sweep line passes a point $p = (x_p, y_p)$ in $L \cup R$; see Figure 2(a). Let L' and L'' be the two active buckets to the left of L , and R' and R'' be the two active buckets to the right of R ; that is, the buckets L'', L', L, R, R', R'' are consecutive in the left-to-right sequence of active buckets. To maintain the invariant, we deactivate buckets L and R and create a new active bucket $X = [x_1, x_3] \times (-\infty, y_p]$; see Figure 2(b). This maintains the invariant because both L' and R' contain at least one active point; otherwise the invariant would have been violated by L' and L or by R and R' before reaching point p . When creating the bucket X , we also create a new cell $C = [x_0, x_4] \times (-\infty, y_p]$ and add it to \mathcal{C} , where x_0 is the left boundary of L'' and x_4 is the right boundary of R'' ; see Figure 2(c).

To prove that the set \mathcal{C} of cells we obtain using this procedure has the desired properties, first note that every bucket contains exactly K points. Every cell in \mathcal{C} contains the active points from five active buckets and, thus, contains at most $5K$ points. The number of cells we create is equal to the number of buckets we create

during the sweep. To bound this number, observe that we create N/K buckets initially, all of which are active. Every time we create a new active bucket, two buckets become inactive. Thus, the number of active buckets decreases by one every time we create a new bucket, and we can repeat this only $N/K - 1$ times before we are left with only one active bucket. This shows that the total number of buckets we create is $2N/K - 1$.

So far we have shown that \mathcal{C} has $O(N/K)$ cells containing $O(K)$ points each. It remains to prove the covering properties of these cells. So consider a three-sided range $q = [a, b] \times (-\infty, c]$, and assume q contains T points. First assume $T \leq K$ and consider the time when the sweep line is at the y -coordinate $Y = c$. Since two consecutive buckets that are active at this time contain more than K active points, the x -range of $[a, b]$ can span at most one active bucket X . This implies that q can be covered using the three buckets L , X , and R , where L and R are the two active buckets adjacent to X . Now assume w.l.o.g. that L was created after X and R . Then X and R were active when L was created, and the cell $C \in \mathcal{C}$ created along with L covers the x -ranges of L , X , and R . Since the top boundary of C is the same as that of L and, thus, is above the line $Y = c$, this implies that C covers q .

For the case $T > K$, consider again the time when the sweep line is at the y -coordinate $Y = c$. As before, since two consecutive active buckets contain more than K active points, the x -range of q can span at most $2T/K$ active buckets, that is, q can be covered using at most $2 + 2T/K$ active buckets. For each such bucket X , the cell in \mathcal{C} created along with X includes X . Thus, q can be covered using at most $2 + 2T/K$ cells in \mathcal{C} .