Range Selection and Median: Tight Cell Probe Lower Bounds and Adaptive Data Structures

Allan Grønlund Jørgensen Kasper Green Larsen*

MADALGO†, Department of Computer Science,
Aarhus University, Denmark
E-mail: {jallan,larsen}@cs.au.dk

Abstract

Range selection is the problem of preprocessing an input array A of n unique integers, such that given a query (i, j, k), one can report the k'th smallest integer in the subarray $A[i], A[i+1], \ldots, A[j]$. In this paper we consider static data structures in the word-RAM for range selection and several natural special cases thereof.

The first special case is known as *range median*, which arises when k is fixed to $\lfloor (j-i+1)/2 \rfloor$. The second case, denoted *prefix selection*, arises when i is fixed to 0. Finally, we also consider the *bounded rank prefix selection* problem and the *fixed rank range selection* problem. In the former, data structures must support prefix selection queries under the assumption that $k \leq \kappa$ for some value $\kappa \leq n$ given at construction time, while in the latter, data structures must support range selection queries where k is fixed beforehand for all queries. We prove cell probe lower bounds for range selection, prefix selection and range median, stating that any data structure that uses S words of space needs $\Omega(\log n / \log(Sw/n))$ time to answer a query. In particular, any data structure that uses $n \log^{O(1)} n$ space needs $\Omega(\log n / \log \log n)$ time to answer a query, and any data structure that supports queries in constant time, needs $n^{1+\Omega(1)}$ space. For data structures that uses $n \log^{O(1)} n$ space this matches the best known upper bound.

Additionally, we present a linear space data structure that supports range selection queries in $O(\log k / \log \log n + \log \log n)$ time. Finally, we prove that any data structure that uses S space, needs $\Omega(\log \kappa / \log(Sw/n))$ time to answer a bounded rank prefix selection query and $\Omega(\log k / \log(Sw/n))$ time to answer a fixed rank range selection query. This shows that our data structure is optimal except for small values of k.

1 Introduction

Range selection is the problem of preprocessing an input array A of n unique integers, such that given a query (i, j, k), one can report the k'th smallest integer in the subarray $A[i], A[i+1], \ldots, A[j]$. In this paper we consider both range selection and several natural special cases thereof. The first special case is known as *range median*, which arises when k is fixed to $\lfloor (j-i+1)/2 \rfloor$. The second case, denoted *prefix selection*, arises when i is fixed to 0. These problems have obvious application in statistical analysis, and have been studied extensively in the last few years, see e.g. [5, 16, 18, 21, 27, 28, 4, 15, 7]. Actually, prefix selection is the inverse of two-dimensional dominance counting which has also been studied extensively; see e.g. [24, 20, 8]. Given an input point set in the plane, a dominance counting query (x, y) returns the number of points in the two sided rectangle $(-\infty, x] \times (-\infty, y]$; a prefix selection query (x, k) returns how large y must be for the two-sided rectangle to contain k points. Finally, we also consider the *bounded rank prefix selection* problem and the *fixed rank range selection* problem. In the former, data structures must support prefix selection queries under the assumption that $k \leq \kappa$ for some value $\kappa \leq n$ given at construction time, while in the latter, data structures must support range selection queries where k is fixed beforehand for all queries. For fixed rank range selection we assume $k \leq n/2$, since otherwise we can enforce this restriction by negating all array entries and fixing the query rank to $n - k$.

The main results of this paper are tight lower bounds for range selection, prefix selection and range median. Surprisingly our lower bounds show that all three variants are equally hard. These results naturally leads us to investigate under what conditions one can improve on these bounds. Specifically we provide a new data structure for range selection where the query time is a function of k instead of n, i.e. the data structure is *adaptive* to k. Finally, we prove lower bounds for bounded rank prefix selection and fixed rank range selection. These bounds not only show that our adaptive data structure is optimal except for very small values of k, but moreover they have interesting implications for generalizing range minimum queries.

*Kasper Green Larsen (has previously published as Kasper Dalgaard Larsen) is a recipient of the Google Europe Fellowship in Search and Information Retrieval, and this research is also supported in part by this Google Fellowship.

†Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

1.1 Previous Results. All upper bound results are in the word-RAM model of computation. In the word-RAM model the memory consists of words of w bits and we can access any word in constant time. Standard word operations like arithmetic, shifts, and logical operations can be performed in constant time. We use the standard assumption in the word-RAM that $w = \Theta(\log n)$. The lower bounds we consider are all in the cell probe model. This is the word-RAM where the complexity of an algorithm is the number of memory cells the algorithm accesses. All other computations are free. Clearly lower bounds in the cell probe model applies to data structures in the word-RAM.

The upper bounds for range selection and the special cases thereof achieved so far divide naturally into two categories, data structures using near-linear space and with polylogarithmic query time, and data structures using close to n^2 space but with constant query time.

Polynomial Space. A data structure supporting range median queries in constant time using $O(n^2 \log\log n/\log n)$ words of space was presented in [21]. The space bound was later improved to $O(n^2 \log^{(k)} n/\log n)$, where $\log^{(k)} n$ is the logarithm applied k times, for any constant k [27], and then to $O(n^2(\log\log n/\log n)^2)$ space [28]. Finally, a data structure that supports range median queries in expected constant time with expected $O(n^{3/2})$ space, under the assumption that all inputs are equally likely was presented in [16].

Near-Linear Space. In [21], the authors present an $O(n\log^2 n/\log\log n)$ space data structure that answers range median queries in $O(\log n)$ time. A linear space data structure capable of answering range median queries in $O(n^\varepsilon)$ time for any constant $\varepsilon > 0$ was also given. These results were later improved to linear space and $O(\log n)$ query time by data structures in [16, 15], which also supports range selection queries in the same bound. Finally, a linear space data structure with $O(\log n/\log\log n)$ query time for range selection was developed [5], where it was also shown how to augment the range selection data structure to support 2-d dominance counting queries in the same query bound. In fact, the same approach can easily be applied to the data structure of [16], thus, it seems there is a strong connection between these two problems.

Lower Bounds. In the word-RAM there are no known lower bounds for any of the selection problems. However, in [24, 25] a cell probe lower bound was shown for the related problem of 2-d dominance counting. This lower bounds states that any data structure that uses S words of space needs $\Omega(\log n/\log(Sw/n))$ time to answer a query. The two papers proves the lower bound with radically different approaches. Given that the current range selection data structures also support 2-d dominance queries, one might conjecture this lower bound also applies to range selection. However, such a lower bound does not follow from either of the two approaches, and perhaps a completely different strategy to designing range selection data structures could give query bounds below the lower bound for 2-d dominance counting.

1.2 Our Results. We settle the complexity of range selection, prefix selection and range median for near-linear space ($n\log^{O(1)} n$) data structures by providing cell probe lower bounds that match the best previous upper bounds. Surprisingly, our lower bounds show that all three variants are equally hard, thus, nothing is gained by restricting attention to the seemingly easier special cases of range selection. Specifically, we prove that any data structure that uses S space needs $\Omega(\log n/\log(Sw/n))$ time to answer a prefix selection query. In particular, any data structure that uses $n\log^{O(1)} n$ space needs $\Omega(\log n/\log\log n)$ time to answer a prefix selection query, and any data structure that supports prefix selection queries in constant time, needs $n^{1+\Omega(1)}$ space. This proves that there is an inherent separation between near-linear space and polynomial space data structures for prefix selection. It also shows that the linear space range selection data structure from [5] is optimal. Interestingly, we prove our lower bound by using a classic hard instance for orthogonal range queries. Our approach is to consider the data structure problem as a communication game between the query algorithm and the data structure. We prove the same lower bound for range median queries by an elegant reduction from prefix selection.

Given that we have settled the complexity of range selection it is natural to investigate under what conditions one can improve on the tight bounds. We consider data structures that are *adaptive* in the sense that their query time depends on k. This adaptive measure has never been applied to range selection data structures, but it has been considered for more than twenty years for several strongly related problems, such as selection in an unsorted array [13] and selection in a cartesian sum $X + Y$ [14]. We give the first adaptive data structure for range selection. We present a linear space data structure that supports range selection queries in $O(\log k/\log\log n + \log\log n)$ time. Our approach is an interesting new application of shallow cuttings [22] which are usually applied to reporting problems.

An obvious question is whether our upper bounds are the best that we can hope for. We prove that it essentially is. More specifically, for data struc-

Table 1: Overview of results proved in this paper

Problem	Space	Query Time
Prefix selection	S	$\Omega(\log n/\log(Sw/n))$
Range median	S	$\Omega(\log n/\log(Sw/n))$
Bounded rank prefix selection	S	$\Omega(\log \kappa/\log(Sw/n))$
Fixed rank range selection	S	$\Omega(\log k/\log(Sw/n))$
Range selection	$O(n)$	$O(\log k/\log\log n + \log\log n)$

tures that uses S space we prove a lower bound of $\Omega(\log \kappa/\log(Sw/n))$ for bounded rank prefix selection queries. This proves that our adaptive data structure for range selection is optimal for $k = 2^{\Omega(\log^2 \log n)}$, and at most an additive term of $O(\log\log n)$ away from optimal otherwise.

This leads us to study one final intriguing question. For range median queries, k is fixed to a specific function of the query interval. It is not clear what happens if we fix k to some specific value for all queries independent of the query interval, i.e. the fixed rank selection problem. The existence of linear space data structures that support range minimum queries (k fixed to 1) in constant time has been known for many years [19]. Recently there has been extensive work reinvestigating this problem and extending it to two and more dimensions, see [12, 2, 3]. There seems to be no reason to believe that this could not be generalized to give data structures with linear (or near-linear) space and constant query time that works for any fixed k. We prove that data structures using S space needs $\Omega(\log k/\log(Sw/n))$ time to answer a fixed rank range selection query. This implies that for any $k = \log^{\omega(1)} n$, any data structure that supports range selection queries for the k'th smallest element in constant time needs $nk^{\Omega(1)}$ space even when k is fixed for all queries. This final result shows that it is not possible to generalize range minimum data structures to fixed rank range selection.

We have summarized our results in Table 1.

1.3 Preliminaries. Throughout the paper we will often think of the input array for all the problems considered as a point set in rank space, i.e. we have one point $(i, A[i])$ for each element in A, and we assume each $A[i]$ is a unique integer in $[n] = \{0, 1, \ldots, n-1\}$.

2 Lower Bounds

In this section we prove lower bounds for prefix selection, range median, bounded rank prefix selection and fixed rank range selection data structures, in this order.

The idea follows a current trend (or framework) in data structure lower bounds in the cell probe model initiated in [23] and developed to its current state by Pătraşcu and Thorup [26, 25, 24]. The strategy is to define *hard* input sets of size $\Theta(n)$ and hard query sets of $b = n/\log^{O(1)} n$ queries and consider a communication game between two players, Alice and Bob. Alice receives a query set and Bob an input set. Bob builds a data structure on the input he receives and Alice simulates a query algorithm for all of her queries in parallel. Assume that the data structure problem has a solution with S space and query time t in the cell probe model. Then there is a communication protocol where Alice sends $t\log\binom{S}{b} = O(tb\log(S/k))$ bits and Bob sends btw bits. The lower bound then comes from providing non-trivial lower bounds on the amount of communication needed by Alice and/or Bob. This strategy has been successfully applied several times. We know of [25, 24, 26, 17, 29].

2.1 Lower Bounds For Prefix Selection. This lower bound proof follows the strategy of Pătraşcu in [24], where a lower bound is proved for data structures supporting dominance counting queries. It seems natural to use this approach given the relationship to dominance counting. For consistency we have used the same notation where possible. The hard input set is basically a modification of the classic bit-reversal permutation which has been used numerous times before [24, 9, 10, 1]. The query sets are essentially evenly spread queries with uniformly random chosen rank k. Similar to [24], the hardness of the query set is shown by considering linear constraints over the input set imposed by queries and their answers. The main difference lies in the way we force the number of linearly independent constraints to be high.

In this proof we think of the input as a point set in rank space. Our construction will use a parameter B where $n = B^h$. We define an input set I that has between n and $2n$ points and a set Q of n/B^2 queries. For the queries we are only interested in the y-coordinate of the answer.

Input Set. The input set consists of two parts. The first part is a simple diagonal, $D = \{(-i, i-0.5) \mid 1 \leq i \leq n\}$ and the second part is a uniformly at

random chosen subset R of the bit-reversal permutation $P = \{(i, rev(i)) \mid i \in [n]\}$, where $rev(i)$ reverses the bits of i including leading zeroes. The reason for the diagonal will become clear later. The input set becomes $I = D \cup R$. Note that this input is not technically in rank space, but we choose this representation for ease of notation.

Query Set. The query set Q consists of n/B^2 prefix selection queries, $q_i = (x_i, k_i+1)$, where $x_i = iB^2$ and k_i is chosen uniformly at random from $[n]$. We think of each value k_i as a number in base B with h digits, i.e. each k_i is a vector in $[B]^h$.

We split the query set into h levels, one level for each digit. Level ℓ, for $\ell = 0, \ldots, h-1$, consists of B^ℓ buckets, and with the i'th bucket we associate the $B^{h-\ell-2}$ queries with x coordinate in $[iB^{h-\ell}, (i+1)B^{h-\ell} - 1]$.

A query q_i is well separated at level ℓ if for every other query q_j in the same bucket at level ℓ, we have $|k_i - k_j| \geq 2B^{\ell+1}$. A query is well separated if it is well separated on all levels. The set of well separated queries we denote Q^*. The following lemma is (essentially) proved in [24].

LEMMA 2.1. *Assume that $B \geq 20000h$. Then with probability at least 99/100, $|Q^*| \geq 99/100|Q|$.*

Digit Sets. To capture the relative ordering of *close* queries we define *digit sets* Γ_i for $i = 0, \ldots, h-1$. These are similar in spirit to the interleave patterns from [24]. Γ_ℓ contains a set for each of the B^ℓ buckets of level ℓ. Each set stores the x-coordinate and the $h-\ell$ most significant digits of k for each query from Q in the corresponding bucket. Clearly the digit sets encode Q and have the following simple property ($H(\cdot)$ denotes entropy).

LEMMA 2.2. $H(\Gamma_\ell \mid \Gamma_{\ell+1}, \ldots, \Gamma_{h-1}) \leq |Q| \log B$

Proof. The only difference between $\Gamma_{\ell+1}$ and Γ_ℓ is one digit per query, and each digit has at most $\log B$ bits of entropy.

Linear Constraints. Assume that we fix the answer to all queries. Every query $q = (x, k)$ and the corresponding answer y defines a rectangle $(-\infty, x] \times (-\infty, y]$. We say that a point $p \in D \cup P$ (not necessarily in I) is contained in query q if p is contained in the rectangle defined by q. The crucial observation is that the answer to q puts a linear constraint on the input, i.e. $\sum_{p \in (P \cup D) \cap (-\infty, x] \times (-\infty, y]} w_I(p) = k$ where $w_I(p) = 1$ if $p \in I$ and zero otherwise. Thus, a set of queries and a set of corresponding answers define a set of linear constraints on I, and it makes sense to talk about the rank of these linear constraints. Consider the linear constraints as a matrix A, where we associate a column in A for each point in $D \cup P$ and a row in A for each query. We set $A_{i,j}$ to one if the point associated to column j is contained in the query associated with row i and zero otherwise. Clearly rank(A) equals the number of linearly independent constraints imposed on I.

We define $Q[E]$ as the set of all queries that are still possible given an event E. Let E be an event such that $|Q^*| \geq 9/10|Q|$. The important property of our queries and point set is captured in the following lemma.

LEMMA 2.3. *If the answers to all queries in $Q[E]$ are fixed, and there exists an index ℓ where $h/2 < \ell < h-2$ such that $H(\Gamma_\ell \mid E, \Gamma_{\ell+1}, \ldots, \Gamma_{h-1}) \geq 9/10|Q| \log B$ then we get at least $1/85|Q|B^{3/4}$ linearly independent constraints on I.*

Proof. First, we restrict our attention to a carefully chosen subset of queries. This subset is similar to the subset considered in [24]. We include the description of how this subset is chosen for completeness. Then, we prove that this subset imposes the required number of linearly independent constraints on I. This is where our proof differs notably from [24].

From $\Gamma_{\ell+1}$ we learn for each query (x, k) the $h-\ell-1$ most significant digits of k, and we have thus determined an interval of size $B^{\ell+1}$ for k.

We eliminate all queries that are not well separated on level $\ell+1, \ldots, h-1$. Furthermore, if two queries $q_i = (x_i, k_i)$ and $q_j = (x_j, k_j)$ are in the same bucket at level ℓ and the $h-\ell-1$ most significant digits of k_i and k_j are the same we eliminate both queries. Note that if this is the case then $|k_i - k_j| < B^\ell$ and thus they are not well separated at level ℓ. After these two steps, Q^* is intact. Furthermore, we eliminate all queries where the $(h-\ell-1)$'st digit of the x-coordinate is smaller than $B/16$. These are the queries with x-coordinate in the first 1/16'th fraction in its corresponding bucket at level ℓ. We still have at least $4/5|Q|$ queries left.

For every query from Q we have deleted we have lost at most $\log B$ bits of the entropy (we loose one *random* digit). Thus, we have $4/5|Q|$ queries left and the entropy of these is at least $7/10|Q| \log B$. It follows that there are at least $2/5|Q|$ queries that have at least $3/4 \log B$ bits of entropy each and thus for each of these we have at least $B^{3/4}$ choices for the ℓ'th most significant digit. This means that $|Q[E]| \geq 2/5|Q|B^{3/4}$.

Now we lower bound the number of linearly independent constraints imposed by $Q[E]$ when the answer to each query is fixed. Consider the buckets $b_1, b_2, \ldots$ of level ℓ ordered by x-coordinate and look at the constraint matrix A defined from the queries and the fixed answers. Order the columns by x-coordinate of the associated points. This way, the range of x-coordinates associated with each bucket defines a set of contigu-

ous columns in A. Observe that the rows corresponding to queries in b_j can only contain nonzero entries in columns associated with $b_1, \ldots, b_j$ (ignoring the diagonal D). Thus we can lower bound rank(A) by considering the queries bucket-wise and sum up the results.

Consider the bucket b_i. We let P_i be the points from P where the x-coordinate is in the first 1/16'th fraction of the range defined by b_i. We lower bound the rank of the linear constraints induced by the remaining queries from b_i by only considering the points from P_i. Since each remaining query from b_i has an x-coordinate that is larger than the x-coordinates of all points in P_i, the linear constraints depends only on the ordering of the y-coordinates of the points and the answers to the queries. Consider the points from P_i sorted by their y-coordinate. Due to the structure of the bit-reversal permutation, two consecutive points in this list differs by precisely $16B^\ell$ on the y-coordinate.

Sort the remaining queries in b_i by their k value, and pick every 34'th query from this list. Now consider two remaining queries $q_1 = (x_1, k_1)$ and $q_2 = (x_2, k_2)$ with answer y_1 and y_2 respectively. Assume wlog. that $k_1 < k_2$. We show that there is at least one point in P_i that is contained in q_2 but not in q_1. This is true if $y_2 \geq y_1 + 16B^\ell$.

Since each k value is unique on the $h - \ell$ most significant digits and we only use every 34'th query, we have $k_2 - k_1 > 33B^\ell$. There are at most $k_1 + B^{h-\ell} \leq k_1 + B^\ell$ points from I in the rectangle $(-\infty, t] \times (-\infty, y_1]$ where $t = (i+1)B^{h-\ell} - 1$ is the right endpoint of b_i, since there are at most $B^{h-\ell}$ points in b_i and $\ell > h/2$. Since $x_2 \leq t$ there must be at least $32B^\ell$ points contained in q_2 that have y-coordinate larger than y_1. Note that if we did not include the diagonal in I, it could be the case that there were not $32B^\ell$ points left in I with y-coordinate larger than y_1. Since I contains at most 2α points in a y-range of length α we conclude that $y_2 \geq y_1 + 16B^\ell$.

It follows that this imposes at least $1/85B^{3/4}$ linearly independent constraints on I.

Cell Probe Lower Bound. We are now ready to use the general framework and consider a communication game between two players. The lower bound is proved by combining the idea in [24] with Lemma 2.3.

Alice gets a query set Q and Bob an input I chosen uniformly at random as described earlier. Bob builds a data structure for his point set that uses S space, and Alice simulates a query algorithm that uses t cell probes. This gives a protocol where Alice sends $t \log \binom{S}{|Q|} = O(t|Q| \log(S/|Q|)$ bits and Bob sends $O(t|Q|w)$ bits.

We fix each message in the protocol to its most likely value. The inputs that would result in this fixed communication defines a combinatorial rectangle where we let the rows be the possible query sets of Alice and the columns the possible input sets of Bob. Note that this fixes the answers to all queries Alice could still ask since Alice must be able to output the answer to all of the remaining queries without further communication. Denote the rows in this rectangle Φ and the columns Ψ. We define the event E to be $Q \in \Phi$.

We restrict our attention to query sets Q such that $|Q^*| \geq 9/10|Q|$. The information about Q revealed by the communication is $H(Q) - H(Q \mid Q \in \Phi) = O(t|Q| \log(S/|Q|))$ since we have fixed the most likely message.

We now assume $t < \varepsilon \log n / \log(S/|Q|)$ for some constant ε sufficiently small, and derive a contradiction. Since the digit sets encode Q, and we have restricted our attention to Q such that $|Q^*| \geq 9/10|Q|$, we get from Lemma 2.1 that $H(\Gamma_0, \ldots, \Gamma_{h-1}) \geq 98/100|Q| \log n$. We can now write

$$H(\Gamma_0, \ldots, \Gamma_{h-1} \mid E) =$$
$$\sum_{i=0}^{h-1} H(\Gamma_i \mid \Gamma_{i+1}, \ldots, \Gamma_{h-1}, E) \geq$$
$$|Q| \log n - O(t|Q| \log(N/|Q|))$$

This is at least $97/100|Q| \log n$ for ε small enough. We can only apply Lemma 2.3 for levels $h/2 < \ell < h - 2$, thus, we eliminate the rest from consideration by upper bounding the amount of entropy they may contain. The sum of $H(\Gamma_{h-1} \mid E), H(\Gamma_{h-2} \mid \Gamma_{h-1}, E)$ and $H(\Gamma_\ell \mid \Gamma_{\ell+1}, \ldots, \Gamma_{h-1}, E)$ for $\ell = 0, \ldots, \lceil h/2 \rceil$ is at most $(h/2 + 3)|Q| \log B$ bits by Lemma 2.2. Since $h = \log n / \log B$, this is at most $51/100|Q| \log n$ for $h \geq 300$, and we are left with $46/100|Q| \log n$ bits of entropy. By averaging over the remaining terms, there must exist a level $h/2 < \ell < h - 2$ for which $H(\Gamma_\ell \mid \Gamma_{\ell+1}, \ldots, \Gamma_{h-1}, E)$ is at least $92/100|Q| \log B$ and we can apply Lemma 2.3. This imposes $1/85|Q|B^{3/4}$ linearly independent constraints on Bobs input. It can be shown by an encoding argument that this is only possible if $H(I) - H(I \mid I \in \Psi) = \Omega(|Q|B^{3/4})$. Since we fixed the most likely message, $H(I) - H(I \mid I \in \Psi) = O(t|Q|w)$ by the bound on Bobs communication. B is still an unspecified parameter and by setting $B = \max\{20000h, (tw)^2\}$ we get a contradiction to our assumption that $t < \varepsilon \log n / \log(S/|Q|)$. Since $h, t = O(\log n) = O(w)$, $B = \log^{O(1)} n$ we get the following theorem.

THEOREM 2.1. *Any data structure that uses S space on an input of size n needs $\Omega(\log n / \log(Sw/n))$ time to answer a prefix range selection query.*

2.2 Reducing Prefix Selection to Range Median. For this reduction, we consider the input as an array. Given an input array I to the prefix selection problem, we provide an input array M for the range median problem such that any prefix selection query in I can be answered by one range median query in M. Let n be the size of I, and set

$$M = [\underbrace{\infty, \ldots, \infty}_{2n \text{ times}}, \underbrace{-\infty, \ldots, -\infty}_{n \text{ times}}, I[0], \ldots, I[n-1]] .$$

Note that we can easily transform the elements of M into rank space coordinates, since all we need is that $-\infty < I[i] < \infty$ for $i \in [n]$. The output of a prefix query (x, k) in I is equal to the output of the range median query $(3n - 2(\lfloor x/2 \rfloor - k), x + 3n)$ in M if $k \leq x/2$ and the range median query $(n - 2(k - \lfloor x/2 \rfloor), x + 3n)$ in M otherwise.

COROLLARY 2.1. *Any data structure that uses S space for an input of size n needs $\Omega(\log n / \log(Sw/n))$ time to answer a range median query.*

2.3 Lower Bound for Bounded Rank Prefix Selection. For this lower bound we consider the input as a point set in rank space. Given κ, we construct an input point set of size $\Theta(n)$ that consists of $\Theta(n/\kappa)$ disjoint point sets of size $\Theta(\kappa)$ and a set Q of $\Theta(n/B^2)$ queries each constructed as in Section 2.1.

Let I_κ be an input set defined as in Section 2.1 with $n = \kappa$, i.e. a diagonal of size κ and a uniformly random subset of the bit-reversal permutation on κ points. We use n/κ such sets $I_\kappa^1, \ldots, I_\kappa^{n/\kappa}$ where we translate the points in I_κ^i by subtracting $i\kappa$ from each y-coordinate and adding $2i\kappa$ to each x-coordinate. Again, note that we can easily transform the point set into rank space.

As in Section 2.1, the query set Q consists of n/B^2 prefix selection queries, constructed as κ/B^2 queries in each I_κ^i. The j'th query in I_κ^i is $(i2n + jB^2, k+1)$ where k is chosen uniformly at random in $[\kappa]$. Now we can basically copy the proof from Section 2.1. This time there are only $\log_B \kappa$ levels. The buckets of level ℓ is constructed by taking the union of the level ℓ buckets for each I_κ^i each defined as in Section 2.1. Notice that the diagonals included in each I_κ^i ensures that the linear constraints imposed by queries in different sets I_κ^i are linearly independent. The entropy of Q becomes $|Q| \log \kappa$ and we get the following corollary.

COROLLARY 2.2. *Any data structure that uses S space on an input of size n takes $\Omega(\log \kappa / \log(Sw/n))$ time to answer a bounded rank prefix selection query.*

2.4 Lower Bound for Fixed Rank Range Selection. For this lower bound we consider the input as an array. We reduce bounded rank prefix selection to fixed rank range selection with $k = \kappa$. Let I and κ be the input to the bounded rank prefix selection problem. We make a new array M by prepending κ entries to I each with value $-\infty$. The output of a bounded rank prefix selection query (x, k) on I is equal to the output of the fixed rank range selection query $(\kappa - (\kappa - k), x) = (k, x)$ on M.

COROLLARY 2.3. *Any data structure that uses S space for an input of size n needs $\Omega(\log k / \log(Sw/n))$ time to answer a fixed rank range selection query.*

3 Adaptive Data Structure

In this section we describe an adaptive data structure for range selection, that uses $O(n)$ space and supports range selection queries in $O(\log k / \log\log n + \log\log n)$ time. We will think of the input as a point set in rank space. Our data structure use at its core the notion of shallow cuttings [22]. Shallow cuttings have mainly been used for reporting problems, but turns out to have several desirable properties for range selection.

A *shallow cutting for the ℓ-level* of I, or an *ℓ-shallow cutting* for short, is a set R of $O(|I|/\ell)$ rectangles such that for any query q that selects for the $k \leq \ell$ smallest element there exists a rectangle in $r \in R$ such that q can be answered by performing the same query only on the points contained in r. In such a case we say that r *resolves* q. Furthermore, any rectangle in R contains $O(\ell)$ points.

In the following we describe the overall idea behind our data structure and in the next section we fill out the details.

Overall Idea. Let I be the input point set. We construct two predecessor data structures that maps x and y-coordinates to rank space respectively, and a static (non-adaptive) range selection data structure on all the points in I with rank space coordinates. Then, we subdivide I using an ℓ-shallow cutting with $\ell = n^\varepsilon$ where $\varepsilon < 1$ is a constant. The points in each rectangle in this shallow cutting are then stored recursively ($\ell = n^{\varepsilon^i}$ at the i'th level of recursion).

To answer a query for the k'th smallest element we start at the first level. We map the query coordinates to rank space using the predecessor data structures and compare k to ℓ. If $k > \ell$ we query the (non-adaptive) data structure stored for all the points. Otherwise, we find a rectangle in the shallow cutting that resolves q and answer the query recursively. Once the answer has been determined, we use the predecessor data structures on each level to remap the compressed answer into the point it represents on the previous level until we have finally remapped the answer to a point in I.

Note that if $k > \ell$ then the number of points left is at most $k^{\frac{1}{\varepsilon}}$ and thus the query on the non-adaptive data structure takes $O(\log k/\log\log n)$ time. Secondly, the rank space reduction allows us to compress the coordinates in the recursive subdivisions and save space.

For the non-adaptive data structure we use the optimal range selection data structure from [5]. We express their result a little different to expose the parts that we need.

THEOREM 3.1. *([5]) There is a data structure that uses $O(n\log(n)/w)$ words of space and supports range selection queries in $O(\log n/\log w)$ time.*

Furthermore, we make extensive use of predecessor data structures, specifically for rank space reductions, and we need the following theorem.

THEOREM 3.2. *There exists a data structure that uses $O(n\log(n)/w)$ words of space and supports predecessor queries in $O(\log\log n)$ time when the universe is of size $n^{O(1)}$.*

Proof. Take a y-fast [30] tree with bucket size w, and store each bucket in a B-tree [11] with $B = \Theta(w)$. Use table lookups to perform operations on blocks in constant time.

Finally, we need the following recent result by Chan[6].

THEOREM 3.3. *([6]) There exists a linear space data structure that supports point location queries in a rectilinear subdivision in $O(\log\log n)$ time.*

3.1 Adaptive Range Selection. In the following we describe how to construct the ℓ-shallow cuttings we use in our adaptive range selection data structure and how we locate a resolving rectangle.

Shallow Cuttings. Given ℓ and a point set I we construct an ℓ-shallow cutting for range selection as follows. For the shallow cutting we use three-sided rectangles on the form $[x_1, x_2]\times(-\infty, y]$ each containing at most 4ℓ points. We associate a key to each rectangle which we need to find a rectangle that resolves a query. The keys are horizontal line segments (x_1, x_2, y).

Imagine sweeping a horizontal line from $y = -\infty$ to ∞. While doing this, we maintain a sorted set of split points $X = \{X_1, X_2, \ldots\}$ that partition the sweep line into disjoint intervals $(X_1, X_2], (X_2, X_3], \ldots$. Initially, $X = \{-\infty, \infty\}$. As we move the sweep line we encounter points (x, y) from I. For each such point we do the following. Let i be the index of the predecessor of x in X. We consider the rectangle $R = [X_i, X_{i+1}] \times (-\infty, y]$. If this rectangle contains less than 2ℓ points from I we continue with the next point. If it contains exactly 2ℓ points from I we compute the median, m, of the x-coordinates amongst the points from I in R. We then output two overlapping rectangles, $[X_{i-1}, X_{i+1}] \times (-\infty, y]$ with key (X_i, m, y) and $[X_i, X_{i+2}] \times (-\infty, y]$ with key (m, X_{i+1}, y). Furthermore, we split the subinterval of the sweep line $(X_i, X_{i+1}]$ into two by inserting m into X. Notice that this ensures that R never contains more than 2ℓ points.

When all the points from I has been processed we additionally output $|X| - 2$ rectangles $[X_i, X_{i+2}] \times (-\infty, \infty)$ for $i = 1, \ldots, |X|-2$. These rectangles do not have keys.

Locating a Resolving Rectangle. Given a query $q = (x_1, x_2, k)$ where $k < \ell$, it is not immediately clear that the set of rectangles constructed always contain a rectangle that resolves q.

LEMMA 3.1. *Let $q = (x_1, x_2, k)$ be a range selection query. If $k < \ell$, then the shallow cutting contains a rectangle that resolves q.*

Proof. Let $(\bar{x}_1, \bar{x}_2, y)$ be the key (horizontal line segment) contained in the x-slab $[x_1, x_2] \times (-\infty, \infty)$ with minimum y-coordinate, if any exists. Then the rectangle R associated with this key resolves q. To prove this, we show that R contains at least ℓ points between x_1 and x_2 and that the x-range of R contains $[x_1, x_2]$. Let X^R be the set X just before the sweep line encountered the point that triggered the output of R. Now consider how the algorithm processed that point. Wlog. assume that $\bar{x}_2$ is the median x-coordinate computed. Then the x-range of R is $[X^R_{i-1}, X^R_{i+1}]$ and $\bar{x}_2 \leq x_2 < X^R_{i+1}$ since otherwise there would be a key (X^R_i, X^R_{i+1}, y') inside the slab with smaller y-coordinate. Similarly, $X^R_{i-1} < x_1 \leq \bar{x}_1$. Since the algorithm splits $[X^R_i, X^R_{i+1}]$ there is exactly ℓ points from I in $[X^R_i, \bar{x}_2] \times (-\infty, y] \subseteq [x_1, x_2] \times (-\infty, y] \subseteq R$.

If no such key exists, we consider the rectangles that does not have keys. Let X be the final set of split points produced by the algorithm. Since $[x_1, x_2] \times (-\infty, \infty)$ does not contain any keys there can be at most one split point in X between x_1 and x_2. Let X_i be the predecessor of x_1 in X then clearly the (keyless) rectangle $[X_i, X_{i+2}] \times (-\infty, \infty)$ resolves q.

From this discussion it is clear that to find a rectangle resolving q we must find the key with minimum y-coordinate in the x-slab defined by q if any exists, or determine the predecessor of x_1 in X otherwise. We transform the former problem to two-dimensional point location in a rectilinear subdivision, that is, a subdivision of the plane constructed from axis-aligned line segments; We map each key (x_a, x_b, y) to the two-dimensional point (x_a, x_b) with associated weight y.

Then given query $q = (x_1, x_2, k)$ the point with smallest weight in the dominance region $[x_1, \infty) \times (-\infty, x_2]$ is the key we are looking for. From this weighted point set we create a rectilinear subdivision as follows. We sort the points by weights in ascending order. We scan the points and for each point p we do the following. Shoot a vertical ray upwards from p and stop once it hits a previous ray. Shoot a ray leftwards until it hits a previous ray. Then delete all remaining points that are to the left of and above p and continue to the next point.

For each cell in the subdivision we associate the bottom right point. Given a query $q = (x_1, x_2, k)$ we find the cell containing (x_1, x_2) and return the associated point. This is the point with minimum weight in the dominance region $[x_1, \infty) \times (-\infty, x_2]$, and thus the key with minimum y-coordinate in the x-slab defined by q.

Data Structure. We use two data structures. The first data structure answers queries where $k > 2^{\log^3 \log n}$ and the second handles the remaining queries. In the data structure for queries with $k > 2^{\log^3 \log n}$ we construct an $n^{1/16}$-shallow cutting on I and recursively construct $n^{1/16^2}$-shallow cuttings for each rectangle and so forth, until the number of points in each rectangle decreases to $2^{\log^3 \log n}$. We store a non-adaptive data structure and two predecessor data structures for each shallow cutting. Furthermore we store the point location data structure of Theorem 3.3 on the keys defined by the shallow cutting and a predecessor data structure on the elements in the final version of X.

Similarly, for the queries where $k \leq 2^{\log^3 \log n}$ we use an ℓ-shallow cutting where $\ell = 2^{\log^3 \log n}$ in the first step, and $\ell = 2^{\log^3 \log n/16^i}$ in the i'th step (including the same data structures).

Query Algorithm. To answer a range selection query $q = (x_1, x_2, k)$ we first examine the query to determine whether $k > 2^{\log^3 \log n}$. If this is the case we query the first data structure, otherwise we query the second. The queries are answered in the following way in both cases.

First, we map the coordinates of the query to rank space. On each level where $k < \ell$, we find a rectangle that resolves q by querying the point location data structure and the predecessor data structure on the final version of X. We then recurse on the resolving rectangle. When $k \geq \ell$ we query the non-adaptive data structure and remap the output to a point in I.

Analysis. In our ℓ-shallow cutting we have $3n/\ell$ rectangles each containing at most 4ℓ points. We store the n points in the range selection data structure from Theorem 3.1 and the $2n/\ell$ keys in the point location data structure from Lemma 3.3 and the n/ℓ keys from X in the predecessor data structure from Theorem 3.2. Not counting the recursive shallow cuttings, the space needed for an ℓ-shallow cutting on a set of size n is $O(n \log n/w + (n/\ell)^{1+\varepsilon})$. Since $\ell = n^{1/16}$ this is $O(n \log n/w)$ for sufficiently small ε. Thus, the space, $S(n)$, needed for the data structure is bounded by $S(n) \leq 3n^{15/16} S(4n^{1/16}) + O(n \log(n)/w)$ which solves to $O(n)$ for $w = \Theta(\log n)$.

To answer a query we perform one non-adaptive range selection query in a data structure with $O(k^{16})$ elements which takes $O(\log k / \log \log n)$ time. If $k > 2^{\log^3 \log n}$ we also perform $O(\log \log n)$ predecessor and point location queries each taking $O(\log \log n)$ time. Since $k > 2^{\log^3 \log n}$ the total query time becomes $O(\log k / \log \log n)$. If $k < 2^{\log^3 \log n}$ we do $O(\log \log \log n)$ predecessor and point location queries in sets of size at most $O(2^{\log^3 \log n})$. In the first predecessor query, the universe is of size n, while in the remaining queries the universe is of size $O(2^{\log^3 \log n})$. In this case we get a query time of $O(\log k / \log \log n + \log \log n)$.

THEOREM 3.4. *There exists a linear space data structure that supports range selection queries in* $O(\log k / \log \log n + \log \log n)$ *time.*

References

[1] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting in three and higher dimensions. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 149–158, 2009.

[2] M. J. Atallah and H. Yuan. Data structures for range minimum queries in multidimensional arrays. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 150–160, 2010.

[3] M. J. Atallah and H. Yuan. Optimal succinctness for range minimum queries. In *Proceedings of the 9th Latin American Theoretical Informatics Symposium*, pages 150–160, 2010.

[4] P. Bose, E. Kranakis, P. Morin, and Y. Tang. Approximate range mode and range median queries. In *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science*, pages 377–388, 2005.

[5] G. S. Brodal and A. G. Jørgensen. Data structures for range median queries. In *Proceedings of the 20th International Symposium on Algorithms and Computation*, pages 822–831, 2009.

[6] T. M. Chan. Persistent predecessor search and orthogonal point location in the word RAM. In *Proceedings of the 22nd ACM/SIAM Symposium on Discrete Algorithms (SODA)*, 2011. to appear.

[7] T. M. Chan and M. Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proceedings of the 21st ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 161–173, 2010.

[8] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal of Computing*, 17:427–462, 1988.

[9] B. Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM*, 37(2):200–212, 1990.

[10] B. Chazelle. Lower bounds for off-line range searching. *Discrete & Computational Geometry*, 17(1):53–65, 1997.

[11] D. Comer. Ubiquitous B-tree. *ACM Computing Survey*, 11(2):121–137, 1979.

[12] E. D. Demaine, G. M. Landau, and O. Weimann. On cartesian trees and range minimum queries. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming*, pages 341–353, 2009.

[13] R. W. Floyd and R. L. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18(3):165–172, 1975.

[14] G. N. Frederickson and D. B. Johnson. The complexity of selection and ranking in X+Y and matrices with sorted columns. *Journal of Computer and System Sciences*, 24(2):197–208, 1982.

[15] T. Gagie, S. J. Puglisi, and A. Turpin. Range quantile queries: Another virtue of wavelet trees. In *Proceedings of the 16th String Processing and Information Retrieval Symposium*, pages 1–6, 2009.

[16] B. Gfeller and P. Sanders. Towards optimal range medians. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming*, pages 475–486, 2009.

[17] M. Greve, A. G. Jørgensen, K. D. Larsen, and J. Truelsen. Cell probe lower bounds and approximations for range mode. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming*, pages 605–616, 2010.

[18] S. Har-Peled and S. Muthukrishnan. Range medians. In *Proceedings of the 16th Annual European Symposium on Algorithms*, pages 503–514, 2008.

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

[20] J. JáJá, C. W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proceedings of the 15th International Symposium on Algorithms and Computation*, pages 558–568, 2004.

[21] D. Krizanc, P. Morin, and M. H. M. Smid. Range mode and range median queries on lists and trees. *Nordic Journal of Computing*, 12(1):1–17, 2005.

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

[23] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998.

[24] M. Pătraşcu. Lower bounds for 2-dimensional range counting. In *Proceedings of the 39th ACM Symposium on Theory of Computing*, pages 40–46, 2007.

[25] M. Pătraşcu. (Data) STRUCTURES. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 434–443, 2008.

[26] M. Pătraşcu and M. Thorup. Higher lower bounds for near-neighbor and further rich problems. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 646–654, 2006.

[27] H. Petersen. Improved bounds for range mode and range median queries. In *Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science*, pages 418–423, 2008.

[28] H. Petersen and S. Grabowski. Range mode and range median queries in constant time and sub-quadratic space. *Information Processing Letters*, 109(4):225–228, 2008.

[29] C. Sommer, E. Verbin, and W. Yu. Distance oracles for sparse graphs. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 703–712, 2009.

[30] D. E. Willard. Log-logarithmic worst-case range queries are possible in space Theta(n). *Information Processing Letters*, 17(2):81–84, 1983.