Succinct Data Structures for Segments

Philip Bille
Department of Applied Mathematics and Computer Science, Technical University of Denmark, Lyngby, Denmark

Inge Li Gørtz
DTU Compute, Technical University of Denmark, Lyngby, Denmark

Simon R. Tarnow
DTU Compute, Technical University of Denmark, Lyngby, Denmark

Abstract

We consider succinct data structures for representing a set of n horizontal line segments in the plane given in rank space to support *segment access*, *segment selection*, and *segment rank* queries. A segment access query finds the segment (x_1, x_2, y) given its y-coordinate (y-coordinates of the segments are distinct), a segment selection query finds the jth smallest segment (the segment with the jth smallest y-coordinate) among the segments crossing the vertical line for a given x-coordinate, and a segment rank query finds the number of segments crossing the vertical line through x-coordinate i with y-coordinate at most y, for a given x and y. This problem is a central component in compressed data structures for persistent strings supporting random access.

Our main result is a data structure using $2n \lg n + O(n \lg n / \lg \lg n)$ bits of space and $O(\lg n / \lg \lg n)$ query time for all operations. We show that this space bound is optimal up to lower-order terms. We will also show that the query time for segment rank is optimal. The query time for segment selection is also optimal by a previous bound.

To obtain our results, we present a novel segment wavelet tree data structure of independent interest. This structure is inspired by and extends the classic wavelet tree for sequences. This leads to a simple, succinct solution with $O(\log n)$ query times. We then extend this solution to obtain optimal query time. Our space lower bound follows from a simple counting argument, and our lower bound for segment rank is obtained by a reduction from 2-dimensional counting.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Succinct, Data structures, Selection

Digital Object Identifier 10.4230/LIPIcs.CPM.2025.27

Related Version *Previous Version*: https://arxiv.org/abs/2412.04965

Funding *Philip Bille*: Danish Research Council grant DFF-8021-002498.
Inge Li Gørtz: Danish Research Council grant DFF-8021-002498.

1 Introduction

Let $\mathcal{L}$ be a set of n horizontal line segments in *rank space*, that is, the line segments are in the plane $[1, 2n] \times [1, n]$ such that there is exactly one endpoint on each x-coordinate and one segment on each y-coordinate. The *segment representation problem* is to preprocess $\mathcal{L}$ to support the operations:

- segment-access(y): return the segment with y-coordinate y.
- segment-select(i, j): return the y-coordinate of the jth smallest segment (the segment with the jth smallest y-coordinate) among the segments crossing the vertical line through x-coordinate i.
- segment-rank(i, y): return the number of segments crossing the vertical line through x-coordinate i with y-coordinate at most y.

Here, we consider a segment (x_l, x_r, y) to be crossing the vertical line through x-coordinate i iff $x_l \leq i < x_r$. The segment representation problem in which the endpoints are real numbers can be reduced to the rank space variant using standard techniques [7]. Bille and Gørtz [2] considered representing segments to support segment selection queries in connection with compressed data structures for persistent strings. Here, the problem is supporting random access on a set of strings represented by a version tree, where each edge represents a replace, insert, or delete operation on the string represented by the parent node. They showed that this problem can be reduced to answering segment selection queries on horizontal line segments. They gave a data structure supporting segment selection queries using $O(n \lg n)$ bits of space and $O(\lg n / \lg \lg n)$ query time[1]. Furthermore, they showed that $\Omega(\lg n / \lg \lg n)$ time is required to answer segment selection queries for any static data structure using $n \lg^{O(1)} n$ bits of space.

1.1 Results

This paper considers succinct data structures for the segment representation problem. We show the following main result on a standard unit cost word RAM model with logarithmic word size.

▶ **Theorem 1.** *Given a set of n horizontal line segments, we can solve the segment representation problem using $2n \lg n + O(n \lg n / \lg \lg n)$ bits of space and $O(\lg n / \lg \lg n)$ time for all queries.*

Compared to previous results of Bille and Gørtz [2], Theorem 1 improves the space bounds from $O(n \lg n)$ to $2n \lg n + O(n \lg n / \lg \lg n)$. At the same time, we obtain the optimal $O(\lg n / \lg \lg n)$ query time for segment selection and implement the segment rank query in the same time. Furthermore, we show that the space bound of Theorem 1 is optimal up to lower order terms.

▶ **Theorem 2.** *Any data structure representing n horizontal line segments requires at least $2n \lg n - O(n)$ bits.*

Finally, we show that the query time for segment rank is also optimal.

▶ **Theorem 3.** *Any static data structure on n horizontal line segments that uses $n \lg^{O(1)} n$ bits of space needs $\Omega(\frac{\lg n}{\lg \lg n})$ time to support segment rank queries.*

1.2 Techniques

We obtain our results by first considering a novel and simple structure, the *segment wavelet tree*, which may be of independent interest. The segment wavelet tree is inspired by the classical wavelet tree structure of Grossi et al. [9], and builds on the observation that the number of segments crossing a vertical line is the difference between the number of left and right endpoints occurring before said line. In the same manner, as the wavelet tree recursively splits the alphabet in its lower and upper half, the segment wavelet tree recursively splits the segments in the lower and upper half of the plane. Because of this, the segment wavelet tree can be used to search for the jth segment crossing a vertical line. The segment wavelet tree, however, only achieves $O(\lg n)$ query time since it only splits the plane into 2 horizontal bands. In order to speed up the query to $O(\lg n / \lg \lg n)$ we generalize the segment

[1] We denote $\lg n = \log_2 n$.

wavelet tree to the Δ-*ary segment wavelet tree*, which splits the plane into Δ horizontal bands called *slabs*, leading to Theorem 1. Next, we prove the information-theoretical lower bound of representing horizontal line segments by showing that in rank space, we need at least $2n \lg n - O(n)$ bits of space to represent n horizontal line segments. Finally, we prove a matching lower bound for the segment rank query on horizontal line segments by showing that any static solution using $n \lg^{O(1)} n$ bits of space needs $\Omega(\lg n / \lg \lg n)$ query time. To do so, we show a reduction from *2-dimensional dominance counting* [19].

1.3 Related Work

Queries on Intervals. There exists a related problem called the *stabbing-semigroup problem*. The stabbing-semigroup problem is to preprocess a set of n intervals where each interval has an associated weight, such that given an integer x, we can compute the sum of weights of the intervals containing x. Agarwal et al. [1] showed how to solve the stabbing-semigroup problem in $O(n \lg n)$ bits of space and $O(\lg n)$ query time. They also showed how to make the structure dynamic, allowing adding and removing intervals in $O(\lg n)$ time and how it can be adapted to work in external memory. Another related query on intervals is the stabbing-max, which is the problem of finding the interval of maximum weight containing x. Nekrich [18] described a dynamic data structure that answers one-dimensional stabbing-max queries in optimal time $O(\lg n / \lg \lg n)$ using $O(n \lg n)$ bits and allows insertions and deletions of intervals in $O(\lg n)$ time. To the best of our knowledge, neither of these problems has been considered in a succinct setting.

Succinct Data Structures. Many geometric queries on two-dimensional points have been considered in a succinct setting, including orthogonal range reporting and counting [5, 14, 16], point location [3, 11] and data-analysis queries [17] (see also the survey by He [10]). We extend this line of research by considering horizontal line segments in a succinct setting.

2-Dimensional Dominance Counting. The *2-dimensional dominance counting problem* is to preprocess n points, such that given a point (x, y), we can compute the number of points (x', y') where $x' \leq x$ and $y' \leq y$. The $\mathsf{segment\text{-}rank}(i, y)$ can also be viewed as two 2-dimensional dominance counting queries by observing that the number of segments crossing the vertical line through x-coordinate i with y-coordinate at most y is the difference in the number of left and right endpoints dominating (i, y). Bose et al. [4] showed that we can answer 2-dimensional dominance counting queries in $O(\lg n / \lg \lg n)$ time using $n \lg n + o(n \lg n)$ space, when the points are in rank space. Thus, we can achieve the same results for segment rank queries as in Theorem 1 using two 2-dimensional dominance counting structures. However, this leaves out how to answer segment access and selection queries.

Range Selection. The *range selection problem* is to preprocess an array A of n unique integers, such that given a query (i, j, k), one can report the kth smallest integer in the subarray $A[i], A[i+1], \ldots, A[j]$. A slight variation of the range selection problem is the *prefix selection problem*, which fixates $i = 1$ in the query. Due to Jørgensen and Larsen [15], the prefix selection problem can be solved in $O(n \lg n)$ bits and $O(\lg n / \lg \lg n)$ query time with matching lower bounds. Bille and Gørtz [2] showed the similarity between prefix selection and segment selection by proving that $\Omega(\lg n / \lg \lg n)$ time is required to answer segment selection queries for any static data structure using $n \lg^{O(1)} n$ space, by reduction from prefix selection.

1.4 Outline

We present the required preliminaries for our solution in Section 2, and then we describe a simple structure in Section 3 and how we can answer segment rank and select queries with this structure. We then show our structure in Section 4 and how we answer rank and select queries succinctly in $O(\lg n / \lg \lg n)$ time leading to Theorem 1. Finally, we prove the space lower bounds for representing segments in rank space leading to Theorem 2 and prove the lower bounds for segment rank queries leading to Theorem 3 in Section 5.

2 Preliminaries

For a given problem P with $|\mathcal{U}|$ instances, we need $\lceil \lg |\mathcal{U}| \rceil$ bits to distinguish between each instance in the worst case. We say a data structure representing P is *compact* if it uses $O(\lceil \lg |\mathcal{U}| \rceil)$ bits of space and is *succinct* if it uses $\lceil \lg |\mathcal{U}| \rceil + o(\lg |\mathcal{U}|)$ bits of space [13].

2.1 Succinct Representations of Strings

Let $S[1, n]$ be a string of n characters from an alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$ and define the following operations.

- $\mathsf{access}(S, i)$: return $S[i]$.
- $\mathsf{rank}(S, \alpha, i)$: return the number of occurrences of character α in $S[1, i]$.
- $\mathsf{select}(S, \alpha, i)$: return the position in S of the ith occurrence of character α.

For binary strings, we use the following well-known result:

▶ **Lemma 4** ([12]). *We can represent a bit string of length n using $n + o(n)$ bits and support access, rank, and select queries in constant time.*

Wavelet Tree. A *wavelet tree* [9] for a string S is a complete balanced binary tree T with σ leaves. Each node v in T represents the characters in $\Sigma(v) = [a, b] \subseteq \Sigma$. The root represents the full alphabet Σ and for any non-leaf node v with alphabet $\Sigma(v) = [a, b]$ the left child v_0 represents the lower half of $\Sigma(v)$, i.e., $\Sigma(v_0) = [a, a + \lfloor (b - a)/2 \rfloor]$, and the right child v_1 the upper half of $\Sigma(v)$, i.e., $\Sigma(v_1) = [a + \lfloor (b - a)/2 \rfloor + 1, b]$. Furthermore, for any leaf node v, we have $\Sigma(v) = [c, c]$. For a node v let $S(v)$ be S restricted to the characters $\Sigma(v)$. Each internal node v with left child v_0 and right child v_1 stores a bitstring $B(v)$ such that $B(v)[i] = 0$ if $S(v)[i] \in \Sigma(v_0)$ and $B(v)[i] = 1$ otherwise ($S(v)[i] \in \Sigma(v_1)$).

With the bit string $B(v)$ we can track which child of v will contain each element $S(v)[i]$, by observing that child v_b, where $b \in [0, 1]$, will contain element $S(v)[i]$ iff $B(v)[i] = b$. If $B(v)[i] = b$ then $S(v)[i]$ will be stored at $S(v_b)[j]$ where index j is the number of occurrences of b in $B(v)[1, i]$ since each occurrence of b is an element in $S(v)$ that would also be in $S(v_b)$ and would be in the same order as they appear in $S(v)$. This is exactly the rank operation on bit strings, thus $S(v)[i] = S(v_b)[\mathsf{rank}(B(v), b, i)]$. With this property, we can navigate downwards in the wavelet tree. We can also use the bit string $B(v)$ together with select to navigate upwards in the wavelet tree. Let v_b be a child of v. Then the index i that the symbol $S(v_b)[j]$ occurs in $S(v)$ is the index of the jth b in $B(v)$ which is $i = \mathsf{select}(B(v), b, j)$. The following lemma captures these properties:

▶ **Lemma 5.** *Let v be a non-leaf node in a wavelet tree. Given an index $i \in [1, |B(v)|]$ then $j = \mathsf{rank}(B(v), b, i)$ is the greatest index in the bit string of the child v_b such that $\mathsf{select}(B(v), b, j) \leq i$. In particular, if $B(v)[i] = b$ then $\mathsf{select}(B(v), b, j) = i$.*

This follows from the duality of rank and select and the observations made above. Using Lemma 4 to represent the bit strings of each node one can implement a wavelet tree for a sequence $S[1,n]$ over the alphabet Σ using $n\lceil \lg \sigma \rceil + o(n \lg \sigma)$ bits of space and $O(\lg \sigma)$ query time for rank, select, and access [16]. A more recent result shows the following:

▶ **Lemma 6** ([8]). *The wavelet tree for a sequence $S[1,n]$ over the alphabet $\Sigma = \{1,2,\ldots,\sigma\}$ uses $n\lceil \lg \sigma \rceil + o(n)$ bits of space and $O(\lg \sigma)$ query time for rank, select, and access.*

3 Segment Wavelet Tree

Here, we present a simple succinct solution to the segment representation problem with the following bounds.

▶ **Theorem 7.** *Given a set of n horizontal line segments in the plane $[1,2n] \times [1,n]$, we can solve the segment representation problem succinctly in $2n \lg n + O(n)$ bits and $O(\lg n)$ time for all queries.*

We will begin by describing the content of our data structure and then show how we can answer queries using this structure.

3.1 Data Structure

Let $\mathcal{L}$ be a set of n horizontal line segments in rank space. We define the segment wavelet tree as a recursive decomposition of the line segments in $\mathcal{L}$ similar to the wavelet tree. For simplicity, we assume that n is a power of 2. Define $\mathcal{L}[a,b] \subseteq \mathcal{L}$ to be the segments with y-coordinate in $[a,b]$. The *segment wavelet tree* T for $\mathcal{L}$ is a complete balanced binary tree on n leaves. Each node v represents a set of segments $\mathcal{L}(v) = \mathcal{L}[a,b]$. The root r represents $\mathcal{L}(r) = \mathcal{L}[1,n] = \mathcal{L}$. Let v be an internal node representing the segments $\mathcal{L}(v) = \mathcal{L}[a,b]$, and let v_0 and v_1 be the left and right child of v, respectively. Then v_0 represents the segments $\mathcal{L}(v_0) = \mathcal{L}[a, \lfloor (b-a)/2 \rfloor]$ and v_1 represents the segments $\mathcal{L}(v_1) = \mathcal{L}[\lfloor (b-a)/2 \rfloor + 1, b]$. A leaf node v represents a single segment $\mathcal{L}(v) = \mathcal{L}[a,a]$.

We store the segment wavelet tree succinctly as follows. Each internal node v stores two bitstrings $B^L(v)$ and $B^R(v)$ of length $|\mathcal{L}(v)|$.

$$B^L(v)[i] = \begin{cases} 0 & \text{if the } i\text{th segment in } \mathcal{L}(v) \text{ ordered by left endpoint is in } \mathcal{L}(v_0) \\ 1 & \text{otherwise} \end{cases}$$

$$B^R(v)[i] = \begin{cases} 0 & \text{if the } i\text{th segment in } \mathcal{L}(v) \text{ ordered by right endpoint is in } \mathcal{L}(v_0) \\ 1 & \text{otherwise} \end{cases}$$

Furthermore, we store a bitstring $E[1,2n]$ where

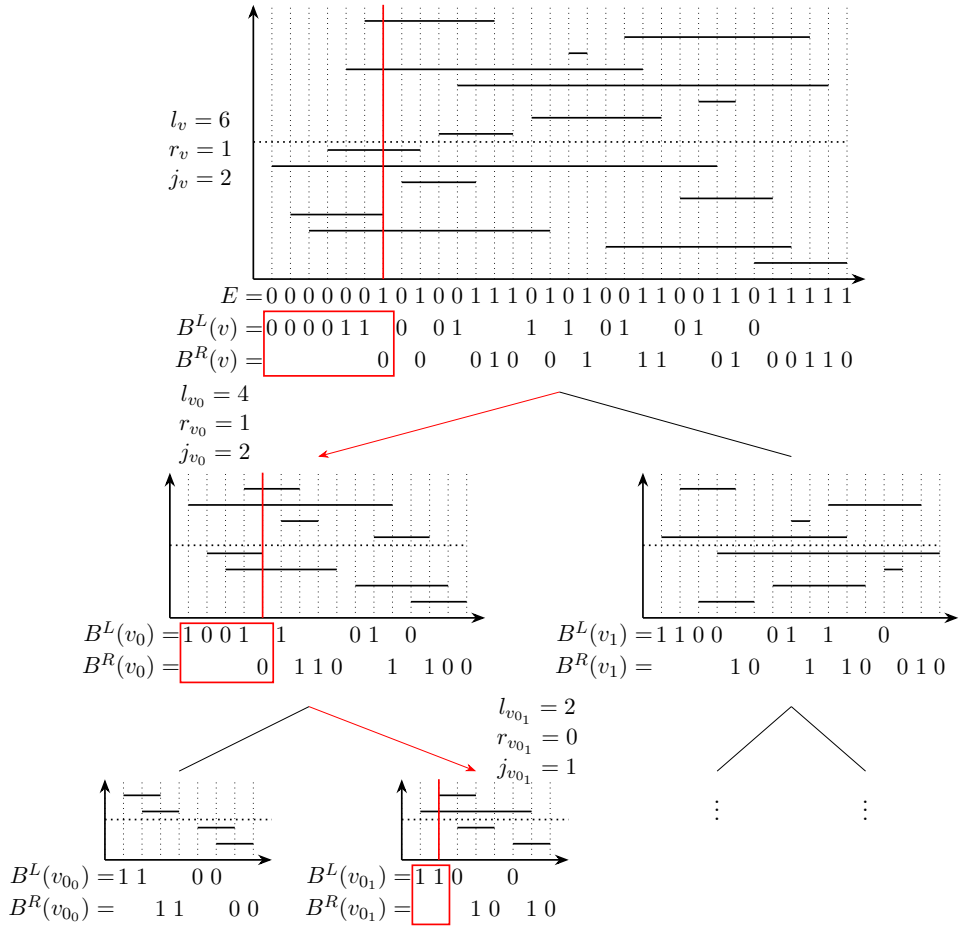
$$E[i] = \begin{cases} 0 & \text{there is a left endpoint with } x\text{-coordinate } i \text{ in } \mathcal{L} \\ 1 & \text{otherwise} \end{cases}$$

See Figure 1 for an example.

Alternatively, one can also view the segment wavelet tree as two superimposed wavelet trees of the strings $Y^L[1,n]$ and $Y^R[1,n]$, where $Y^L[1,n]$ and $Y^R[1,n]$ are the strings of y-coordinates of the left and right endpoints, respectively, ordered by increasing x-coordinate.

To achieve the desired space bounds, we store B^L and B^R as their superimposed wavelet trees Y^L and Y^R according to Lemma 6 and the bitstring E according to Lemma 4.

Analysis. The total length of Y^L and Y^R is $2n$ and the length of E is $2n$. By Lemmas 4 and 6 the total space is $2n\lceil \lg n \rceil + o(n) + 2n + o(n) = 2n \lg n + O(n)$ bits.



$$\mathcal{L} = \{(27,32,1),(19,29,2),(3,16,3),(2,7,4),(23,28,5),(8,12,6),(1,25,7),(4,9,8),(10,14,9),\\ (15,22,10),(24,26,11),(11,31,12),(5,21,13),(17,18,14),(20,30,15),(6,13,16)\}$$

Figure 1 The top 3 levels of the segment wavelet tree of the segments $\mathcal{L}$ and the computed local variables for the query segment-select$(7,2)$, where v is root of the segment wavelet tree. For some of the nodes, the corresponding subproblem is visualized as a 2D plane, where empty columns have been removed. The bitvectors B^L and B^R of each node is horizontally spaced such that each bit vertically aligns with the endpoint it represents. The visited nodes in the query segment-select$(7,2)$ are marked with a red arrow together with the local variables. Furthermore, in the 2D plane of the visited nodes, the vertical line with x-coordinate 7 is highlighted, and the prefix of the bitvectors B^L and B^R that correspond to the endpoints with x-coordinate at most 7 are also highlighted.

3.2 Segment Access Queries

We now show how to answer segment-access queries in $O(\lg n)$ time. To answer a segment-access(y) query, we do a bottom-up traversal of the segment wavelet tree T starting at the yth leaf in the left-to-right order. Let $s = (x_l, x_r, y)$ denote the segment we are searching for. At each node v in the traversal we maintain the following *local variables*:

l_v = the number of segments in $\mathcal{L}(v)$ whose left endpoint has x-coordinate at most x_l.

r_v = the number of segments in $\mathcal{L}(v)$ whose right endpoint has x-coordinate at most x_r.

We perform the traversal as follows. Initially, v is the yth leaf and $l_v = r_v = 1$. Consider a non-root node v with parent v_p and let $b = 0$ if v is the left child of v_p and $b = 1$ otherwise. We compute $l_{v_p} = \mathsf{select}(B^L(v_p), b, l_v)$ and $r_{v_p} = \mathsf{select}(B^R(v_p), b, r_v)$. When we reach the root u we compute $x_l = \mathsf{select}(E, 0, l_u)$ and $x_r = \mathsf{select}(E, 1, r_u)$. Finally, we return (x_l, x_r, y).

Analysis. We use constant time at each node, and we visit one node at each level of the segment wavelet tree. Since the height of T is $O(\lg n)$, the total time is $O(\lg n)$.

Correctness. Let $s = (x_l, x_r, y)$ be the segment with y-coordinate y. We show inductively that l_v and r_v are computed correctly for each v on the path in the bottom-up traversal. When v is the yth leaf we have $\mathcal{L}(v) = \{s\}$ and thus $l_v = 1$ and $r_v = 1$. Consider an internal non-root node v. Assume $s \in \mathcal{L}(v)$ and l_v and r_v are correct. Let v_p be the parent of v and $b = 0$ if v is the left child of v_p and $b = 1$ otherwise. The bitstrings $B^L(v_p)$ and $B^R(v_p)$ are b in every index corresponding to a segment in $\mathcal{L}(v)$. Thus $l_{v_p} = \mathsf{select}(B^L(v_p), b, l_v)$ is the number of segments in $\mathcal{L}(v_p)$ whose left endpoint has x-coordinate at most x_l, and $r_{v_p} = \mathsf{select}(B^R(v_p), b, r_v)$ is the number of segments in $\mathcal{L}(v_p)$ whose right endpoint has x-coordinate at most x_r. Let u be the root. Since E are 0 and 1 in every index corresponding to a left and right segment endpoint, respectively, we have $x_l = \mathsf{select}(E, 0, l_u)$ and $x_r = \mathsf{select}(E, 1, r_u)$.

3.3 Segment Select Queries

We now show how to answer segment-select queries in $O(\lg n)$ time. To answer a segment-select(i, j) query, we do a top-down traversal in the segment wavelet tree T starting at the root and ending in the leaf containing the jth crossing segment of the vertical line at time i.

At each node v with $\mathcal{L}(v) = \mathcal{L}[a, b]$ in the traversal we maintain the following *local variables*:

l_v = the number of segments in $\mathcal{L}(v)$ whose left endpoint has x-coordinate at most i.

r_v = the number of segments in $\mathcal{L}(v)$ whose right endpoint has x-coordinate at most i.

$\bar{j}_v$ = the number of segments in $\mathcal{L}[1, a-1]$ crossing the vertical line at i.

$j_v = j - \bar{j}_v$, such that the j_vth segment in $\mathcal{L}(v)$ crossing the vertical line at i is the jth segment in $\mathcal{L}$ crossing the vertical line at i.

We perform the traversal as follows. Initially, v is the root and we have $r_v = \mathsf{rank}(E, 1, i)$, $l_v = i - r_v, \bar{j}_v = 0$ and $j_v = j$. Consider an internal node v with children v_0 and v_1 and suppose we have computed l_v, $r_v, \bar{j}_v$, and j_v. We first compute the number of segments k in $\mathcal{L}(v_0)$ crossing the vertical line at i as

$$k = \mathsf{rank}(B^L(v), 0, l_v) - \mathsf{rank}(B^R(v), 0, r_v) \ .$$

That is, k is computed as the number of left endpoints of segments in $\mathcal{L}(v_0)$ with x-coordinate at most i subtracted by the number of right endpoints in $\mathcal{L}(v_0)$ with x-coordinate at most i.

We continue the traversal in child v_b, where $b = 0$ if $j_v \leq k$ and $b = 1$ otherwise.

We then compute the local variables for the child v_b as

$$l_{v_b} = \mathsf{rank}(B^L(v), b, l_v)$$
$$r_{v_b} = \mathsf{rank}(B^R(v), b, r_v)$$
$$\bar{j}_{v_b} = \begin{cases} \bar{j}_v & \text{if } b = 0 \\ \bar{j}_v + k & \text{otherwise} \end{cases}$$
$$j_{v_b} = j - \bar{j}_{v_b}$$

When v is a leaf and $\mathcal{L}(v) = \mathcal{L}[a, a]$ we return a.

Analysis. We use constant time at each node and visit one node at each level of the segment wavelet tree. Since the height of T is $O(\lg n)$, the total time is $O(\lg n)$.

Correctness. Let s be the jth smallest segment crossing the vertical line at i. We show inductively, that $s \in \mathcal{L}(v)$ and l_v, r_v, $\bar{j}_v$, and j_v are computed correctly for each v on the path in the top-down traversal.

When v is the root we have $s \in \mathcal{L}(v) = \mathcal{L}$ and $\bar{j}_v = 0$ and $j_v = j$. By definition, we have $r_v = \mathsf{rank}(E, 1, i)$ is the number of segments in $\mathcal{L}$ whose right endpoint has x-coordinate at most i. Then, $l_v = i - r_v = \mathsf{rank}(E, 0, i)$ is the number of segments in $\mathcal{L}$ whose left endpoint has x-coordinate at most i.

Consider an internal non-root node v. Assume $s \in \mathcal{L}(v)$ and that l_v, r_v, $\bar{j}_v$, and j_v are correct. Let v_b be the child of v computed by the algorithm. The bitstrings $B^L(v)$ and $B^R(v)$ are 0 in every index corresponding to a segment in $\mathcal{L}(v_0)$. Thus $k = \mathsf{rank}(B^L(v), 0, l_v) - \mathsf{rank}(B^R(v), 0, r_v)$ is the number of segments crossing the vertical line at i in $\mathcal{L}(v_0)$. The set $\mathcal{L}(v_0)$ contains the lower half of the segments in $\mathcal{L}(v)$. Thus if $j_v \leq k$ then $s \in \mathcal{L}(v_0)$, $\bar{j}_{v_0} = \bar{j}_v$ and $j_{v_0} = j_v = j - \bar{j}_{v_0}$. Otherwise, $s \in \mathcal{L}(v_1)$, $\bar{j}_{v_1} = \bar{j}_v + k$ and $j_{v_1} = j_v - k = j - \bar{j}_{v_1}$. It follows that we continue the traversal in the correct child v_b. By definition of $B^L(v)$ and $B^R(v)$ it follows that l_{v_b}, and r_{v_b} are computed correctly.

3.4 Segment Rank Queries

We now show how to answer segment-rank queries in $O(\lg n)$ time. To answer a segment-rank(i, y) we perform a top-down traversal in the segment wavelet tree T starting at the root and ending in the leaf v such that $\mathcal{L}(v) = \mathcal{L}[y, y]$. At each node v in the traversal we compute the local variables l_v, r_v and $\bar{j}_v$ in the same manner as in Section 3.3. We perform the traversal as follows. Consider an internal node v with $\mathcal{L}(v) = \mathcal{L}[a, b]$ and children v_0 and v_1, we continue the traversal in child v_b, where $b = 0$ if $y \leq \lfloor (b - a)/2 \rfloor$ and $b = 1$ otherwise. When v is a leaf we return $\bar{j}_v + (l_v - r_v)$.

Analysis. We use constant time at each node, and we visit one node at each level of the segment wavelet tree. Since the height of T is $O(\lg n)$, the total time is $O(\lg n)$.

Correctness. The correctness follows immediately from the correctness argument in Section 3.3 and the definition of l_v, r_v and $\bar{j}_v$.

In summary, we have shown Theorem 7.

4 Generalized Segment Wavelet Tree

Here, we generalize the solution in Section 3 to a tree of out-degree $\Delta = \lceil \lg^\epsilon n \rceil$, where $0 < \epsilon < 1$. To increase the out-degree to Δ, we first consider the required data structure to partition the segments into Δ horizontal bands called *slabs*. Afterward, we will describe the content of our data structure and show how we can answer queries using this structure.

4.1 Succinct Slab Representation

Let $\mathcal{L}$ be a set of n horizontal line segments partitioned into $\Delta = \lceil \lg^\epsilon n \rceil$ slabs of approximately equal size, where $0 < \epsilon < 1$. The *slab representation problem* is to preprocess $\mathcal{L}$ to support the operations:

- slab-select(v, i, j): return the slab k containing the jth segment in $\mathcal{L}(v)$ according to increasing y-coordinate among the segments crossing the vertical line through x-coordinate i.
- slab-rank(v, i, j): return the number of segments in $\mathcal{L}(v)$ crossing the vertical line through x-coordinate i in slabs $[1, j]$.
- endpoint-select(v, k, i): return the x-coordinate of the ith endpoint in $\mathcal{L}(v)$ in the kth slab according to increasing x-coordinate among the segments.
- endpoint-rank(v, k, i): return the number of endpoints in $\mathcal{L}(v)$ in the kth slab who has a x-coordinate of at most i.

In [2] they show that slab-select and slab-rank can be done in $O(n \lg \lg^\epsilon n)$ space and constant query time using $O(n)$ preprocessing time. We show how to modify this result and the analysis to achieve $2n \lg \lg^\epsilon n + O(n)$ bits of space while maintaining constant query time.

▶ **Lemma 8.** *Given a set of n horizontal line segments, partitioned into $\Delta = \lceil \lg^\epsilon n \rceil$ horizontal slabs for $0 < \epsilon < 1$, we can solve the slab representation problem in $2n \lg \lg^\epsilon n + O(n)$ bits of space and $O(1)$ time for all queries.*

Proof. In [2] they partition the sequence of segment endpoints into blocks, which are further partitioned into cells. Their data structure consists of the following components.

1. A predecessor data structure for each block.
2. The first column of each cell.
3. The sequence of slab indices each endpoint belongs to, ordered by increasing x-coordinate.
4. A global table for tabulating queries inside the cells.

We modify the block width to be $\lceil \lg^\lambda n \rceil \lceil \lg n \rceil$ instead of $\lceil \lg^\epsilon n \rceil \lceil \lg n \rceil$, for another parameter $\epsilon < \lambda < 1$. This also sets the cell width to $\lceil \lg^\lambda n \rceil$. Plugging into their analysis, this implies the following space bounds.

1. The predecessor structure uses $O(\lg^\epsilon n \lg n)$ space for each block. The number of blocks is $\frac{2n}{\lceil \lg^\lambda n \rceil \lceil \lg n \rceil}$, hence the space required is $O(2n \frac{\lg^\epsilon n}{\lg^\lambda n}) = o(n)$.
2. Each entry of the first column of a cell can be encoded in $O(\lg \lg^\epsilon n)$. The combined height of all cells is $2n \frac{\lceil \lg^\epsilon n \rceil}{\lceil \lg^\lambda n \rceil}$ and thus the combined space is $2n \frac{\lceil \lg^\epsilon n \rceil}{\lceil \lg^\lambda n \rceil} \cdot O(\lg \lg^\epsilon n) = o(n)$.
3. The sequence contains $2n$ endpoints, and each endpoint can be encoded in $\lg \lg^\epsilon n + O(1)$ bits, thus the entire sequence uses $2n \lg \lg^\epsilon n + O(n)$ space.
4. The global table has size $2^{O((\lg^\lambda n + \lg^\epsilon n) \cdot \lg \lg n)} \cdot O(\lg \lg^\epsilon n) = 2^{O((\lg^\lambda n + \lg^\epsilon n) \cdot \lg \lg n)} = o(n)$.

In total, the data structure uses $2n \lg \lg^\epsilon n + O(n)$ space and the same $O(n)$ preprocessing time. Using standard techniques [6], we can also support select and rank on the sequence of endpoints in constant time using $O(n)$ extra space. Since the sequence of endpoints is ordered by increasing x-coordinate select and rank are equivalent with endpoint-select and endpoint-rank, respectively. ◀

4.2 Data Structure

Let $\mathcal{L}$ be a set of n horizontal line segments in rank space. We define the Δ-ary segment wavelet tree as a recursive decomposition of the line segments in $\mathcal{L}$ similar to the Δ-ary wavelet tree. For simplicity, we assume that n is a power of Δ. Define $\mathcal{L}[a, b] \subseteq \mathcal{L}$ to be the segments with y-coordinate in $[a, b]$. The *Δ-ary segment wavelet tree* T for $\mathcal{L}$ is a complete balanced $\Delta = \lceil \lg^\epsilon n \rceil$ tree on n leaves. Each node v represents a set of segments $\mathcal{L}(v) = \mathcal{L}[a, b]$. The root r represents $\mathcal{L}(r) = \mathcal{L}[1, n] = \mathcal{L}$. Let v be an internal node representing the segments $\mathcal{L}(v) = \mathcal{L}[a, b]$, and let $v_0, \ldots, v_{\Delta-1}$ be the children of v. Then v_i represents the segments $\mathcal{L}(v_i) = \mathcal{L}[a + \lfloor (1 + b - a)/\Delta \cdot i \rfloor, a + \lfloor (1 + b - a)/\Delta \cdot (i + 1) \rfloor - 1]$. A leaf node v represents a single segment $\mathcal{L}(v) = \mathcal{L}[a, a]$. We store the Δ-ary segment wavelet tree succinctly as follows. Intuitively, each internal node v stores the structure of Lemma 8, but to achieve the desired space complexity, we instead, for each level in the Δ-ary segment wavelet tree, concatenate the segments of the nodes on that level into one single structure of Lemma 8, as in [16]. Since there is exactly one segment of each y coordinate, we can easily maintain the indices into the single structure corresponding to each node. When n is not a power of Δ, we pack the segments such that all levels in T are complete except possibly the lowest, which is filled from the left. We then store the path along with indices to the rightmost leaf on the lowest level. This allows us to compute the indices into the single structure of each level.

Analysis. Each level of the tree contains all the segments and thus uses $2n \lg \lg^\epsilon n + O(n)$ bits of space by Lemma 8. Since the height of this tree is $\log_\Delta n = \lg n / \lg \lg^\epsilon n$ the entire data structure uses $2n \lg n + O(n \lg n / \lg \lg n) = 2n \lg n + O(n \lg n / \lg \lg n)$ bits of space and $O(n \lg n / \lg \lg n)$ preprocessing time.

4.3 Segment Access Queries

We now show how to answer segment-access queries in $O(\lg n / \lg \lg n)$ time. To answer a segment-access(y) query, we first compute the path to the yth leaf in the left-to-right order. We can trivially compute the path by performing a top-down traversal of the Δ-ary segment wavelet tree T starting at the root and ending in the yth leaf. We then do a bottom-up traversal of the Δ-ary segment wavelet tree T starting at the yth leaf. Let $s = (x_l, x_r, y)$ denote the segment we are searching for. At each node v in the traversal we maintain the following *local variables*:

l_v = the number of endpoints in $\mathcal{L}(v)$ who has x-coordinate at most x_l.

r_v = the number of endpoints in $\mathcal{L}(v)$ who has x-coordinate at most x_r.

We perform the traversal as follows. Initially, v is the yth leaf and $l_v = 1$ and $r_v = 2$. Consider a non-root node v with parent v_p and let v be the kth child of v_p. We compute $l_{v_p} =$ endpoint-select(v_p, k, l_v) and $r_{v_p} =$ endpoint-select(v_p, k, r_v). When we reach the root u we return (l_u, r_u, y).

Analysis. We use constant time at each node and visit one node at each level of the Δ-ary segment wavelet tree. Since the height of T is $O(\lg_\Delta n) = O(\lg n / \lg \lg n)$, the total time is $O(\lg n / \lg \lg n)$.

Correctness. Let $s = (x_l, x_r, y)$ be the segment with y-coordinate y. We show inductively that l_v and r_v are computed correctly for each v on the path in the bottom-up traversal. When v is the yth leaf we have $\mathcal{L}(v) = \{s\}$ and thus $l_v = 1$ and $r_v = 2$. Consider an internal

non-root node v. Assume $s \in \mathcal{L}(v)$ and l_v and r_v are correct. Let v_p be the parent of v and let v be the ith child of v_p. The segments in $\mathcal{L}(v)$ are in slab i of v_p. Thus by the definition of endpoint-select $l_{v_p} = \mathsf{endpoint\text{-}select}(v_p, b, l_v)$ is the number of endpoints in $\mathcal{L}(v_p)$ who has x-coordinate at most x_l and $r_{v_p} = \mathsf{endpoint\text{-}select}(v_p, b, r_v)$ is the number of endpoints in $\mathcal{L}(v_p)$ who has x-coordinate at most x_r. When we arrive at the root u we have $x_l = l_u$ and $x_r = r_u$.

4.4 Segment Select Queries

We now show how to answer segment-select queries in $O(\lg n / \lg\lg n)$ time. To answer a $\mathsf{segment\text{-}select}(i, j)$ query, we do a top-down traversal in the Δ-ary segment wavelet tree T starting at the root and ending in the leaf containing the jth crossing segment of the vertical line at time i.

At each node v with $\mathcal{L}(v) = \mathcal{L}[a, b]$ in the traversal we maintain the following *local variables*:

i_v =the number of endpoints in $\mathcal{L}(v)$ who has x-coordinate at most i.

$\bar{j}_v$ =the number of segments in $\mathcal{L}[1, a-1]$ crossing the vertical line at i.

j_v =$j - \bar{j}_v$, such that the j_vth segment in $\mathcal{L}(v)$ crossing the vertical line at i is the jth segment in $\mathcal{L}$ crossing the vertical line at i.

We perform the traversal as follows. Initially, v is the root and we have $i_v = i$, $\bar{j}_v = 0$ and $j_v = j$. Consider an internal node v with children $v_0, \ldots, v_{\Delta-1}$ and suppose we have computed l_v, $\bar{j}_v$, and j_v. We compute the child v_k containing the j_vth segment crossing the vertical line at i_v as

$$k = \mathsf{slab\text{-}select}(v, i_v, j_v)$$

We then compute the local variables for the child v_k as

$$i_{v_k} = \mathsf{endpoint\text{-}rank}(v, k, i_v)$$
$$\bar{j}_{v_k} = \mathsf{slab\text{-}rank}(v, i_v, k-1) + \bar{j}_v$$
$$j_{v_k} = j - \bar{j}_{v_k}$$

When v is a leaf and $\mathcal{L}(v) = \mathcal{L}[a, a]$ we return a.

Analysis. We use constant time at each node and visit one node at each level of the Δ-ary segment wavelet tree. Since the height of T is $O(\lg n / \lg\lg n)$, the total time is $O(\lg n / \lg\lg n)$.

Correctness. Let s be the jth smallest segment crossing the vertical line at i. We show inductively that $s \in \mathcal{L}(v)$ and i_v, $\bar{j}_v$, and j_v are computed correctly for each v on the path in the top-down traversal.

When v is the root we have $s \in \mathcal{L}(v) = \mathcal{L}$ and $\bar{j}_v = 0$ and $j_v = j$. By definition, we have $i_v = i$ is the number of endpoints in $\mathcal{L}$ with x-coordinate at most i.

Consider an internal non-root node v. Assume $s \in \mathcal{L}(v)$ and that i_v, $\bar{j}_v$, and j_v are correct. Let v_k be the child of v computed by the algorithm. The segments in $\mathcal{L}(v_k)$ are in slab k of v. By the definition of slab-select the j_vth segment crossing the vertical line at i_v is $k = \mathsf{slab\text{-}rank}(v, i_v, k-1)$. Furthermore by the definition $\mathsf{slab\text{-}rank}(v, i_v, k-1)$ is the number of segments crossing the vertical line at i in $\mathcal{L}(v_0) \cup \ldots \cup \mathcal{L}(v_{k-1})$ and $\mathsf{endpoint\text{-}rank}(v, k, i_v)$ is the number of endpoints in $\mathcal{L}(v_k)$ who has x-coordinate at most i. Hence $i_{v_k} = \mathsf{endpoint\text{-}rank}(v, k, i_v)$ and $\bar{j}_{v_k} = \mathsf{slab\text{-}rank}(v, i_v, k-1) + \bar{j}_v$.

4.5 Segment Rank Queries

We now show how to answer segment-rank queries in $O(\lg n/\lg n \lg n)$ time. To answer a segment-rank(i, y), we perform a top-down traversal in the segment wavelet tree T starting at the root and ending in the leaf v such that $\mathcal{L}(v) = \mathcal{L}[y, y]$. At each node v with $\mathcal{L}(v) = \mathcal{L}[a, b]$ in the traversal, we compute the local variables i_v, and $\bar{j}_v$ in the same manner as in Section 4.4. We perform the traversal as follows. Consider an internal node v with $\mathcal{L}(v) = \mathcal{L}[a, b]$ and children $v_0, \ldots, v_{\Delta-1}$. we continue the traversal in child v_k, where $k = \lfloor (y-a)/((1+b-a)/\Delta) \rfloor$. When v is a leaf we return $\bar{j}_v + 1$ if $i_v = 1$ and $\bar{j}_v$ otherwise.

Analysis. We use constant time at each node and visit one node at each level of the segment wavelet tree. Since the height of T is $O(\lg n/\lg\lg n)$, the total time is $O(\lg n/\lg\lg n)$.

Correctness. The correctness follows immediately from the correctness argument in Section 4.4 and the definition of i_v, $\bar{j}_v$ and slab-rank.

In summary, we achieve Theorem 1.

5 Lower Bounds

5.1 Horizontal Line Segments in Rank Space

Here, we show Theorem 2. Let $\mathcal{L}$ be a set of n horizontal line segments in rank space on the plane $[1, 2n] \times [1, n]$ such that there is exactly one endpoint on each x-coordinate and one segment on each y-coordinate. If we only consider the x-coordinates of the left and right endpoint of each segment, then the number of ways to arrange n pairs of endpoints is the number of ways that we can pair $2n$ elements.

$$\prod_{i=1}^{n}(2i-1) = \frac{(2n)!}{2^n n!}$$

Since each segment has a unique y-coordinate, the number of ways the segments can be arranged on the y-axis is $n!$. Thus, the number of ways to arrange n segments in rank space $[1, 2n] \times [1, n]$ is $\frac{(2n)!}{2^n}$. Thus to distinguish between each instance we need $\left\lceil \lg \frac{(2n)!}{2^n} \right\rceil = 2n \lg n - O(n)$ bits. In summary, we have shown Theorem 2.

5.2 Segment Rank in Rank Space

Here, we show Theorem 3. We reduce from *2-dimensional dominance counting*. The 2-dimensional dominance counting problem is to preprocess n points $(x_1, y_1), \ldots, (x_n, y_n)$ from rank space, such that given a point (x, y) compute the number of points (x_i, y_i) such that $x_i \leq x$ and $y_i \leq y$.

▶ **Lemma 9** ([19]). *Any static data structure on n points that uses $n \lg^{O(1)} n$ bits of space requires $\Omega(\frac{\lg n}{\lg\lg n})$ time to support dominance counting queries.*

Given n points $(x_1, y_1), \ldots, (x_n, y_n)$ to the 2-dimensional dominance counting problem, we construct an instance of the segment representation problem. We assume wlog. that these n points are in rank space on the plane $[1, n] \times [1, n]$ such that there is exactly one point on each x and y coordinate. To see why this assumption is acceptable to establish the lower bound, we refer to the discussion by Pătraşcu [19]. We construct the n segments $\mathcal{L}$ as follows: For each point (x_i, y_i) we construct the corresponding segment such that the left endpoint

is (x_i, y_i) and the right endpoint is $(n + x_i, y_i)$ for $1 \leq i \leq n$. For a given query point (x, y), we count the number of points (x_i, y_i) such that $x_i \leq x$ and $y_i \leq y$ by performing the query segment-rank(x, y). Since no segment ends before x-coordinate n, the number of segments crossing the vertical line through x-coordinate x with y-coordinate in $[1, y]$ are the segments with left endpoints (x_i, y_i) such that $x_i \leq x$ and $y_i \leq y$. In summary, we have shown Theorem 3.

References

1. Pankaj K. Agarwal, Lars Arge, Haim Kaplan, Eyal Molad, Robert Endre Tarjan, and Ke Yi. An optimal dynamic data structure for stabbing-semigroup queries. *SIAM J. Comput.*, 41(1):104–127, 2012. doi:10.1137/10078791X.
2. Philip Bille and Inge Li Gørtz. Random access in persistent strings and segment selection. *Theory Comput. Syst.*, 67(4):694–713, 2023. doi:10.1007/s00224-022-10109-5.
3. Prosenjit Bose, Eric Y. Chen, Meng He, Anil Maheshwari, and Pat Morin. Succinct geometric indexes supporting point location queries. *ACM Trans. Algorithms*, 8(2):10:1–10:26, 2012. doi:10.1145/2151171.2151173.
4. Prosenjit Bose, Meng He, Anil Maheshwari, and Pat Morin. Succinct orthogonal range search structures on a grid with applications to text indexing. In Frank K. H. A. Dehne, Marina L. Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth, editors, *Algorithms and Data Structures, 11th International Symposium, WADS 2009, Banff, Canada, August 21-23, 2009. Proceedings*, volume 5664 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2009. doi:10.1007/978-3-642-03367-4_9.
5. Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988. doi:10.1137/0217026.
6. Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. Compressed representations of sequences and full-text indexes. *ACM Trans. Algorithms*, 3(2):20, 2007. doi:10.1145/1240233.1240243.
7. Harold N Gabow, Jon Louis Bentley, and Robert E Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 135–143, 1984.
8. Alexander Golynski, Roberto Grossi, Ankur Gupta, Rajeev Raman, and S. Srinivasa Rao. On the size of succinct indices. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, volume 4698 of *Lecture Notes in Computer Science*, pages 371–382. Springer, 2007. doi:10.1007/978-3-540-75520-3_34.
9. Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 841–850. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644250.
10. Meng He. Succinct and implicit data structures for computational geometry. In Andrej Brodnik, Alejandro López-Ortiz, Venkatesh Raman, and Alfredo Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, volume 8066 of *Lecture Notes in Computer Science*, pages 216–235. Springer, 2013. doi:10.1007/978-3-642-40273-9_15.
11. Meng He, Patrick K. Nicholson, and Norbert Zeh. A space-efficient framework for dynamic point location. In Kun-Mao Chao, Tsan-sheng Hsu, and Der-Tsai Lee, editors, *Algorithms and Computation - 23rd International Symposium, ISAAC 2012, Taipei, Taiwan, December 19-21, 2012. Proceedings*, volume 7676 of *Lecture Notes in Computer Science*, pages 548–557. Springer, 2012. doi:10.1007/978-3-642-35261-4_57.
12. Guy Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October*

- *1 November 1989*, pages 549–554. IEEE Computer Society, 1989. doi:10.1109/SFCS.1989.63533.

13 Guy Joseph Jacobson. *Succinct Static Data Structures*. PhD thesis, Carnegie Mellon University, USA, 1988. AAI8918056.

14 Joseph F. JáJá, Christian Worm Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In Rudolf Fleischer and Gerhard Trippen, editors, *Algorithms and Computation, 15th International Symposium, ISAAC 2004, Hong Kong, China, December 20-22, 2004, Proceedings*, volume 3341 of *Lecture Notes in Computer Science*, pages 558–568. Springer, 2004. doi:10.1007/978-3-540-30551-4_49.

15 Allan Grønlund Jørgensen and Kasper Green Larsen. Range selection and median: Tight cell probe lower bounds and adaptive data structures. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 805–813. SIAM, 2011. doi:10.1137/1.9781611973082.63.

16 Veli Mäkinen and Gonzalo Navarro. Rank and select revisited and extended. *Theor. Comput. Sci.*, 387(3):332–347, 2007. doi:10.1016/j.tcs.2007.07.013.

17 Gonzalo Navarro, Yakov Nekrich, and Luís Manuel Silveira Russo. Space-efficient data-analysis queries on grids. *Theor. Comput. Sci.*, 482:60–72, 2013. doi:10.1016/j.tcs.2012.11.031.

18 Yakov Nekrich. A dynamic stabbing-max data structure with sub-logarithmic query time. *CoRR*, abs/1109.3890, 2011. arXiv:1109.3890.

19 Mihai Puatracscu. Lower bounds for 2-dimensional range counting. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 40–46. ACM, 2007. doi:10.1145/1250790.1250797.