Fast Preprocessing for Optimal Orthogonal Range Reporting and Range Successor with Applications to Text Indexing

Younan Gao
Faculty of Computer Science, Dalhousie University, Halifax, Canada
yn803382@dal.ca

Meng He
Faculty of Computer Science, Dalhousie University, Halifax, Canada
mhe@cs.dal.ca

Yakov Nekrich
Department of Computer Science, Michigan Technological University, Houghton, MI, USA
yakov.nekrich@googlemail.com

Abstract

Under the word RAM model, we design three data structures that can be constructed in $O(n\sqrt{\lg n})$ time over n points in an $n \times n$ grid. The first data structure is an $O(n \lg^{\epsilon} n)$-word structure supporting orthogonal range reporting in $O(\lg \lg n + k)$ time, where k denotes output size and ϵ is an arbitrarily small constant. The second is an $O(n \lg \lg n)$-word structure supporting orthogonal range successor in $O(\lg \lg n)$ time, while the third is an $O(n \lg^{\epsilon} n)$-word structure supporting sorted range reporting in $O(\lg \lg n + k)$ time. The query times of these data structures are optimal when the space costs must be within $O(n \operatorname{polylog} n)$ words. Their exact space bounds match those of the best known results achieving the same query times, and the $O(n\sqrt{\lg n})$ construction time beats the previous bounds on preprocessing. Previously, among 2d range search structures, only the orthogonal range counting structure of Chan and Pătraşcu (SODA 2010) and the linear space, $O(\lg^{\epsilon} n)$ query time structure for orthogonal range successor by Belazzougui and Puglisi (SODA 2016) can be built in the same $O(n\sqrt{\lg n})$ time. Hence our work is the first that achieve the same preprocessing time for optimal orthogonal range reporting and range successor. We also apply our results to improve the construction time of text indexes.

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

Keywords and phrases orthogonal range search, geometric data structures, orthogonal range reporting, orthogonal range successor, sorted range reporting, text indexing, word RAM

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.54

Related Version A full version of the paper is available at `http://arxiv.org/abs/2006.11978`.

1 Introduction

Two dimensional orthogonal range search problems have been studied intensively in the communities of computational geometry, data structures and databases. The goal of these problems is to maintain a set, N, of points on the plane in a data structure such that one can efficiently compute aggregate information about the points contained in an axis-aligned query rectangle Q. Among these problems, *orthogonal range counting* and *orthogonal range reporting* are perhaps the most fundamental; the former counts the number of points contained in $N \cap Q$ while the latter reports them. Another well-known problem is *orthogonal range successor*, which asks for the point in $N \cap Q$ with the smallest x- or y-coordinate. Range

counting, reporting and successor have many applications including text indexing [23, 8, 6, 25], Lempel-Ziv factorization [4] and consensus trees in phylogenetics [18], to name a few. See [22] for a survey on the connection between text indexing and various range searching techniques.

Most work on orthogonal range searching [13, 17, 11, 28, 32] focuses on achieving the best tradeoffs between query time and space, and preprocessing time is often neglected. However, the preprocessing time of a data structure matters when it is used as a building block of an algorithm processing plain data, as the total running time includes that needed to build the structure. Furthermore, an orthogonal range search structures with fast construction time are preferred when preprocessing huge amounts of data, e.g., when used as components of text indexes built upon large data sets from search engines and bioinformatics applications. The work of Chan and Pătraşcu [12] is the first that breaks the $O(n \lg n)$ bound on the construction time of 2d orthogonal range counting structures; they designed an $O(n)$-word structure with $O(\lg n / \lg \lg n)$ query time that can be built in $O(n\sqrt{\lg n})$ time. Their ideas were further extended to design an $O(n \lg \sigma / \sqrt{\lg n})$-time algorithm to build a binary wavelet trees over a string of length n drawn from $[\sigma]$ [26, 2][1], which is a key data structure used in succinct text indexes. More recently, Belazzougui and Puglisi [4] showed how to construct, in $O(n\sqrt{\lg n})$ time, an $O(n)$-word data structure supporting range successor in $O(\lg^{\epsilon} n)$ time, and applied this algorithm to achieve new results on Lempel-Ziv parsing.

The previous work on constructing orthogonal range search structures in $O(n\sqrt{\lg n})$ time focuses on linear space data structures. To achieve optimal query time for 2d orthogonal range reporting and range successor using near-linear space, however, the best tradeoffs under the word RAM model requires superlinear space [11, 32]. The increased space costs are needed to encode more information, posing new challenges to fast construction. We thus investigate the problem of designing data structures with optimal query times for range reporting and range successor that can be built in $O(n\sqrt{\lg n})$ time, while matching the space costs of the best known solutions. We also consider a closely related problem called *sorted range reporting* [28] to achieve similar goals. In this problem, we report all points in $N \cap Q$ in a sorted order along either x- or y-axis. The query time should depend on the number of points actually reported even if the procedure is ended early by user.

Previous Work. The research on 2d orthogonal range reporting has a long history [30, 13, 1, 17, 27, 19, 9, 11]. Researchers have achieved three best tradeoffs between query time and space costs under the word RAM model; we follow the state of the art and assume that the input points are in rank space. The solution with optimal query time of $O(\lg \lg n + k)$ and space cost of $O(n \lg^{\epsilon} n)$ words is due to Alstrup et al. [1], while the best linear-space solution is designed by Chan et al [11] which answers a query in $O((1+k) \lg^{\epsilon} n)$ time, where k is the output size and ϵ is an arbitrarily small constant. Chan et al. also proposed an $O(\lg \lg n)$-word structure with $O((1+k) \lg \lg n)$ query time and another tradeoff matching that of Alstrup et al. [1].

The 2d orthogonal range successor problem was also studied extensively. After a series of work [21, 20, 15, 14, 31], Nekrich and Navarro [28] gave two solutions to this problem; the first uses $O(n)$ words and answers a query in $O(\lg^{\epsilon} n)$ time, while the second uses $O(n \lg \lg n)$ words to answer a query in $O((\lg \lg n)^2)$ time. Zhou [32] decreased the query time of the latter to $O(\lg \lg n)$ without increasing space costs. By definition, a solution to orthogonal range successor can be used to answer sorted range reporting queries. Furthermore, Nekrich and Navarro [28] also designed a data structure using $O(n \lg^{\epsilon} n)$ words to support sorted range

[1] In this paper, $[\sigma]$ denotes $\{0, 1, \ldots, \sigma - 1\}$.

reporting in $O(\lg\lg n + k)$ time. Hence, the best three time-space tradeoffs for the original 2d orthogonal range reporting problem has also been achieved for the sorted version. The optimality of the $O(\lg\lg n + k)$ query time for orthogonal range reporting and the $O(\lg\lg n)$ query time for orthogonal range successor when no more than $O(n \operatorname{polylog} n)$ space can be used is established by a lower bound on range emptiness [29].

Alstrup et al. [1] claimed that their structure for optimal orthogonal range reporting can be constructed in $O(n \lg n)$ expected time. Even though preprocessing times are not given in [11, 28, 32], straightforward analyses reveal that the other data structures we surveyed here can be built in $O(n \lg n)$ worst-case time (Bille and Gørtz [6] also claimed that the preprocessing time of the $O(n \lg\lg n)$-word structure of Chan et al. [11] is $O(n \lg n)$). Hence, when faster preprocessing time is needed in their solution to Lempel-Ziv decomposition, Belazzougui and Puglisi [4] had to design a new linear-space data structure for orthogonal range successor with $O(n\sqrt{\lg n})$ preprocessing time and $O(\lg^{\epsilon} n)$ query time. No attempts have been published to achieve similar preprocessing times for other tradeoffs.

Our Results. Under the word RAM model, we design the following three data structures that can be constructed in $O(n\sqrt{\lg n})$ time over n points in an $n \times n$ grid:

- An $O(n \lg^{\epsilon} n)$-word structure supporting orthogonal range reporting in $O(\lg\lg n + k)$ time, where k denotes the output size and ϵ is an arbitrarily small constant;
- An $O(n \lg\lg n)$-word structure supporting orthogonal range successor in $O(\lg\lg n)$ time;
- An $O(n \lg^{\epsilon} n)$-word structure supporting sorted range reporting in $O(\lg\lg n + k)$ time.

The query times of these structures are optimal when space costs must be within $O(n \operatorname{polylog} n)$ words. Their exact space bounds match those of the best known results achieving the same query times, and the $O(n\sqrt{\lg n})$ construction time beats the previous bounds on preprocessing. Note that even though our third result implies the first one, our data structure for the first is much simpler. In addition, our results can be used to improve the construction time of text indexes. For a text string T of length n over alphabet $[\sigma]$, we design

- A text index of $O(n \lg\sigma \lg^{\epsilon} n)$ bits that can be constructed in $O(n \lg\sigma/\sqrt{\lg n})$ time and can report the $\mathtt{occ}$ occurrences of a pattern of length p in time $O(p/\log_{\sigma} n + \log_{\sigma} n \lg\lg n + \mathtt{occ})$, where ϵ is any small positive constant. This improves one result of Munro et al. [25] who designed the first text indexes with both sublinear construction time and query time for small σ; for the same time-space tradeoff, their preprocessing time is $O(n \lg\sigma \lg^{\epsilon} n)$.
- A text index of $O(n \lg^{1+\epsilon} n)$ bits for any constant $\epsilon > 0$ built in $O(n\sqrt{\lg n})$ time that supports position-restricted substring search [23] in $O(p/\log_{\sigma} n + \lg p + \lg\lg\sigma + \mathtt{occ})$ time. Previous indexes with similar query performance require $O(n \lg n)$ construction time.

Overview of Our Approach. We first discuss why some obvious approaches will not work. The modern approach of Chan et al [11] for orthogonal range reporting is based on a problem called ball inheritance which they defined over range trees. This solution is well-known for its simplicity, and by choosing different parameters in their approach to ball inheritance, they obtain all three best known tradeoffs. One natural idea is to redesign the structures stored at range tree nodes to use bit packing to speed up construction. However, even though we have achieved construction time matching the state of the art for these structures, it is still not enough to construct the data structures for the tradeoffs of ball inheritance that we need quickly enough. Another idea is to tune the parameters in the approach of Belazzougui and Puglisi [4], hoping to obtain the tradeoffs that we aim for, as they already showed how to construct in $O(n\sqrt{\lg n})$ time a linear space, $O((k+1) \lg^{\epsilon} n)$ query

time structure for orthogonal range reporting. Their solution uses many trees grouped into $O(\lg^\epsilon n)$ levels of granularity. If we borrow ideas from [11] and set parameters to achieve different tradeoffs, we would use $O(1/\epsilon)$ or $O(\lg\lg n)$ levels of granularity. However, to return a point in the answer, their query algorithm would perform operations requiring $O(\lg\lg n)$ time at each level of granularity. Thus, at best, the former would give an $O(n\lg^\epsilon n)$-word structure with $O((k+1)\lg\lg n)$ query time and the latter an $O(n\lg\lg n)$-word structure with $O((k+1)(\lg\lg n)^2)$ query time. Either solution is inferior to best known tradeoffs. This however is fine in the original solution, as the total cost of spending $O(\lg\lg n)$ time at each of the $O(\lg^\epsilon n)$ levels is bounded by $O(\lg^{\epsilon'} n)$ for any $\epsilon' > \epsilon$.

We thus design new approaches. For optimal orthogonal range reporting, our overall strategy is to perform two levels of reductions, making it sufficient to solve ball inheritance in special cases with fast preprocessing time. More specifically, we first use a generalized wavelet tree and range minimum/maximum structures to reduce the problem in the general case to the special case in which the points are from a $2^{\sqrt{\lg n}} \times n'$ (narrow) grid, where $n' \leq n$. In this reduction, we need only support ball inheritance over a wavelet tree with high fanout. We further reduce the problem over points in a narrow grid to that over a (small) grid of size at most $2^{\sqrt{\lg n}} \times 2^{2\sqrt{\lg n}}$. This is done by grouping points and selecting representatives from each group, so that previous results with slower preprocessing time can be used over a smaller set of representatives. Finally, over the small grid, we solve ball inheritance when the coordinates of each point can be encoded in $O(\sqrt{\lg n})$ bits. The ball inheritance structures in both special cases can be built quickly by redesigning components with fast preprocessing, though the second case requires a twist to the approach of Chan et al [11]. Our solutions to optimal range successor and sorted range reporting are based on similar strategies, though we preform more levels of reductions.

In the main body of this paper, we describe our data structures for optimal range reporting and successor, while those for optimal sorted range reporting are deferred to the full version of this paper.

2 Preliminaries

In this section, we describe and sometimes extend the previous results used in this paper. The proofs omitted from this section can be found in the full version of this paper.

Notation. We adopt the word RAM model with word size $w = \Theta(\lg n)$ bits, where n denotes the size of the given data. Our complete solutions use several sets of homogeneous components. We present a lemma to bound the costs of each different type of components, which is then applied over the entire set of these components to calculate the total cost. The size, n', of the data that each component represents may be less than n which is the input size of the entire problem, so when the cost of constructing the component is bounded by a function of the form $f(n')/\operatorname{polylog}(n)$ to take advantage of the word size, we keep both n' and n in the lemma statement, as commonly done in previous work on similar topics. In this case, the construction algorithm usually uses a universal table of $o(n)$ bits, whose content solely depends on the value of n, and hence can be constructed once in $o(n)$ time and used for all data structure components of the same type. Thus unless otherwise stated, these lemmas assume the existence of such a table without stating so explicitly in the lemma statements, and we define and analyze the table in the proof. This also applies to algorithms that manipulate sequences of size n'. Occasionally the query algorithms of a data structure may need a universal table as well, and we explicitly state it if this is the case.

We say a sequence $A \in [\sigma]^n$ is in *packed* form if the bits of its elements are concatenated and stored in as few words as possible. Thus, when packed, A occupies $\lceil n\lceil \lg \sigma \rceil / w \rceil$ words.

Generalized Wavelet Trees. Given a sequence $A[0..n-1]$ drawn from alphabet $[\sigma]$, a d-ary generalized wavelet tree [24] T_d over A is a balanced tree in which each internal node has d children, where $2 \leq d \leq \sigma$. For simplicity, assume that σ is a power of d. Each node of T_d then represents a range of alphabet symbols defined as follows: At the leaf level, the i-th leaf from the left represents the integer range $[i, i]$ for each $i \in [0..\sigma - 1]$. The range represented by an internal node is the union of the ranges represented by its children. Hence the root represents $[0, \sigma - 1]$, and T_d is a complete tree having $\log_d \sigma + 1$ levels. Each node u is further associated with a subsequence, $A(u)$, of A, in which $A(u)[i]$ stores the i-th entry in A that is in the range represented by u. Thus the root is associated with the entire sequence A. To save storage, $A[u]$ is not stored explicitly in [24]. Instead, each internal node u stores a sequence $S(u)$ of integers in $[d]$, where $S(u)[i] = j$ if $A(u)[i]$ is within the range represented by the jth child of u. All the $S(u)$'s built for internal nodes occupy $O(n \lg \sigma)$ bits in total.

Generalized wavelet trees share fundamental ideas with range trees but are more suitable for compact data structures over sequences which may contain duplicate values. When we use them in this paper, we sometimes explicitly store $A(u)$ for each node u, and may even associate with u an additional array $I(u)$ in which $I(u)[i]$ stores the index of $A(u)[i]$ in the original sequence A. We call $A(u)$ the *value array* of u, and $I(u)$ the *index array*. In this paper, if we construct value and/or index arrays for each node, we explicitly state so. If not, it implies that we build a wavelet tree in which each node u is associated with $S(u)$ only. Furthermore, unless otherwise specified, we apply the standard pointer-based implementation to represent the tree structure of a wavelet tree, which is preprocessed in time linear to the number of tree nodes such that the lowest common ancestor of any two nodes can be located in $O(1)$ time [5]. We also number the levels of the tree incrementally starting from the root level, which is level 0. We have the following two lemmas on constructing wavelet trees:

▶ **Lemma 1.** *Let $A[0..n'-1]$ be a packed sequence drawn from alphabet $[\sigma]$ and $I[0..n'-1]$ be a packed sequence in which $I[i] = i$ for each $i \in [0..n'-1]$, where $n' \leq n$ and $\sigma \leq 2^{O(\sqrt{\lg n})}$. Given A and I as input, a d-ary wavelet tree over A with value and index arrays in packed form can be constructed in $O(n' \lg \sigma(\lg n' + \lg \sigma)/\lg n + \sigma)$ time, where d is an arbitrary power of 2 with $2 \leq d \leq \sigma$. If index arrays are not constructed, the construction time can be lowered to $O(n' \lg^2 \sigma / \lg n + \sigma)$; this bound still applies when neither value nor index arrays are built.*

▶ **Lemma 2.** *Let $A[0..n-1]$ be a sequence drawn from alphabet $[\sigma]$. A d-ary wavelet tree over A with value and index arrays can be built in $O(n \lg \sigma / \lg d)$ time where $2 \leq d \leq \sigma$.*

A sequence $A[0..n-1]$ drawn from $[\sigma]$ can be viewed as a point set $N = \{(A[i], i) | 0 \leq i \leq n-1\}$. Let T be a d-ary wavelet tree constructed over A. Then *ball inheritance* [11] can be defined over T which asks for the support of these operations: i) `point`(v, i), which returns the point $(A(v)[i], I(v)[i])$ in N for an arbitrary node v in T and an integer i; and ii) `noderange`(c, d, v), which, given a range $[c, d]$ and a node v of T, finds the range $[c_v, d_v]$ such that $I(v)[i] \in [c, d]$ iff $i \in [c_v, d_v]$. If we store the value and index arrays explicitly, it is trivial to support these operations, but the space cost is high. To save space, we only store $S(v)$ for each node v and design auxiliary structures. The following lemma presents previous results:

▶ **Lemma 3** ([11, Theorem 2.1], [10, Lemma 2.3])**.** *A generalized wavelet tree over a sequence $A[0..n-1]$ drawn from $[\sigma]$ can be augmented with ball inheritance data structure in $O(n \lg n f(\sigma))$ bits to support* `point` *in $O(g(\sigma))$ time and* `noderange` *in $O(g(\sigma) + \lg \lg n)$ time, where (a) $f(\sigma) = O(1)$ and $g(\sigma) = O(\lg^\epsilon \sigma)$; (b) $f(\sigma) = O(\lg \lg \sigma)$ and $g(\sigma) = O(\lg \lg \sigma)$; or (c) $f(\sigma) = O(\lg^\epsilon \sigma)$ and $g(\sigma) = O(1)$.*

Data Structures for `rank` and `select`. Given a sequence A drawn from alphabet $[\sigma]$, a $\texttt{rank}_c(A, i)$ operation computes the number of elements equal to c in $A[0..i]$, where $c \in [\sigma]$, while a $\texttt{select}_c(A, i)$ returns the index of the entry of A containing the i-th occurrence of c. We have the following two lemmas on building `rank`/`select` structures.

▶ **Lemma 4.** *Let $A[0..n'-1]$ be a packed sequence drawn from alphabet $[\sigma]$, where $n' \leq n$ and $\sigma = O(\text{polylog}\, n)$. A data structure of $n'\lceil \lg \sigma \rceil + o(n' \lg \sigma)$ bits supporting* `rank` *in $O(1)$ time can be constructed in $O(n' \lg^2 \sigma / \lg n + \sigma)$ time.*

▶ **Lemma 5** ([2, Lemma 2.1])**.** *Given a packed bit sequence $B[0..n-1]$, a systematic data structure occupying $o(n)$ extra bits can be constructed in $O(n/\lg n)$ time, which supports* `rank` *and* `select` *in constant time.*

In the above lemma, a data structure is *systematic* if it requires the input data to be stored verbatim along with the additional information for answering queries. A restricted version of `rank` is called *partial rank*; a partial rank operation, $\texttt{rank}'(A, i)$, computes the number of elements equal to $A[j]$ in $A[0..j]$. The following lemma presents a solution to supporting $\texttt{rank}'$, which is an easy extension of [3, Lemma 3.5].

▶ **Lemma 6.** *Given a sequence $A[0..n-1]$ drawn from alphabet $[\sigma]$, a data structure of $O(n \lg \sigma)$ bits can be constructed in $O(n + \sigma)$ time, which supports* $\texttt{rank}'$ *in constant time.*

Range Minimum/Maximum. Given a sequence A of n integers, a range minimum/maximum query $\texttt{rmq}(i, j)$/$\texttt{rMq}(i, j)$ with $i \leq j$ returns the position of a minimum/maximum element in the subsequence $A[i..j]$. Fischer and Heun [16] considered this problem:

▶ **Lemma 7** ([16])**.** *Given an array A of n integers, a data structure of $O(n)$ bits can be constructed in $O(n)$ time, which answers* `rmq`/`rMq` *in $O(1)$ time without accessing A.*

We further build an auxiliary structure upon a packed sequence A under the *indexing model*: after the the data structure is built, A itself need not be stored verbatim; to answer a query, it suffices to provide an operator that can retrieve any element in A.

▶ **Lemma 8.** *Let $A[0..n'-1]$ be a packed sequence drawn from alphabet $[\sigma]$, where $\sigma \leq 2^{\sqrt{\lg n}}$ and $n' \leq n$. There is a data structure using $O(n' \lg \lg n)$ extra bits constructed in $O(n' \lg \sigma / \lg n)$ time, which answers* `rmq`/`rMq` *in $O(1)$ time and $O(1)$ accesses to the elements of A. The query procedure uses a universal table of $o(n)$ bits.*

3 Fast Construction of $\texttt{rank}'$ Query Structures

In this section we focus on how to efficiently construct data structures for $\texttt{rank}'$ queries over a sequence $A[0..n'-1]$ drawn from alphabet $[\sigma]$, where $n' \leq n$ and $\sigma \leq 2^{\sqrt{\lg n}}$. This is needed to solve ball inheritance in a special case. Lemma 4 already solves this problem when $\sigma \leq \lg n$, so we assume $\lg n < \sigma \leq 2^{\sqrt{\lg n}}$ in the rest of this section.

In our solution, we conceptually divide sequence A into chunks of length σ. For simplicity, assume that n' is a multiple of σ. Let A_k denote the kth chunk, where $0 \leq k \leq n'/\sigma - 1$. For each $c \in [0, \sigma - 1]$, we define the following data structures:

- A bitvector $B_c = 1^{\texttt{rank}_c(A_0,\sigma)}01^{\texttt{rank}_c(A_1,\sigma)}0 \ldots 1^{\texttt{rank}_c(A_{n'/\sigma-1},\sigma)}0$, which encodes the number of occurrences of symbol c in each chunk in unary. B_c is represented using Lemma 5 to support `rank` and `select` in constant time.
- A sequence $P_c[0..n'/\sigma - 1]$, in which $P_c[i] = \texttt{rank}'(A_i, c)$ for each $i \in [0, n'/\sigma - 1]$, i.e., $P_c[i]$ stores the answer to a partial rank query performed locally within A_i at position c.

Note that we have one B_c for each alphabet symbol c, while we have one P_c for each relative position c in the chunks of A. We have the following lemma on supporting queries using these data structures, with a space analysis.

▶ **Lemma 9.** *The data structures in this section occupy $n' \lg \sigma + o(n' \lg \sigma)$ extra bits and support* `rank'` *in $O(1)$ time and $O(1)$ accesses to elements of A.*

Proof. In B_c, each 1 bit corresponds to an occurrence of symbol c in A, while each 0 corresponds to a chunk. Thus, these bit vectors have n' 1s and $n'/\sigma \times \sigma = n'$ 0s in total. Therefore, the lengths of all these bit vectors sum up to $2n'$. By Lemma 5, $o(n')$ bits are needed to augment them to support `rank` and `select`. As each chunk has σ elements, encoding an entry of each P_c requires $\lceil \lg \sigma \rceil$ bits. Thus $P_0, \ldots, P_{\sigma-1}$ occupy $n' \lceil \lg \sigma \rceil$ bits in total. The total space usage of all the data structures in this section is therefore $2n' + o(n') + n' \lceil \lg \sigma \rceil$ bits, which is $n' \lg \sigma + o(n' \lg \sigma)$ when $\sigma > \lg n$.

A query $\texttt{rank}'(A, j)$ can be answered as follows:

$$\texttt{rank}'(A, j) = select_0(B_c, t) - (t-1) + P_\tau[t], where\ \tau = j \bmod \sigma, t = \lfloor \frac{j}{\sigma} \rfloor, and\ c = A[j]$$

As the `select` query over B_c takes constant time, answering $\texttt{rank}'(A, j)$ requires $O(1)$ time and a single access to A. ◀

Next, we consider how to construct the sequences B_c's efficiently.

▶ **Lemma 10.** *Bitvectors $B_0, B_1, \ldots, B_{\sigma-1}$ can be constructed in $O(n' \lg^2 \sigma / \lg n + \sigma)$ time.*

Proof. We first construct a sequence $M[0..n' + n'/\sigma - 1]$ in which each element is encoded in $\lceil \lg \sigma \rceil + 1$ bits. In M, n' elements are *regular elements*, and the rest are *boundary elements* each of which is an integer whose binary expression simply consists of $\lceil \lg \sigma \rceil + 1$ 0-bits. M is divided into n'/σ chunks, and each chunks contains σ regular elements followed by a boundary element. The subsequence of the σ regular elements in the i-th chunk can be obtained by appending a 1-bit to the end of the binary expression of each element in A_k.

Next we show how to create M efficiently with the help of a universal table U. This table has an entry for each possible pair (D, t), where D is a sequence of length $b = \lfloor \frac{\lg n}{2\lceil \lg \sigma \rceil} \rfloor$ drawn from $[\sigma]$ and t is an integer in $[0, b]$. If $t = 0$, this entry stores a sequence of length b which is obtained by appending a 1-bit to the end of the binary expression of each element in D. Otherwise, this entry stores a sequence of length $b+1$ consisting of three sections: the first section is obtained by appending a 1-bit to the end of the binary expression of each of the first t elements in D, the second section is a boundary element, and the third section is obtained by appending a 1-bit to the end of the binary expression of each of the last $b - t$ elements in D. As there are at most $n^{1/2}$ possible sequences of length b drawn from σ and t has $b+1$ possible values, U has at most $n^{1/2}(b+1)$ entries. Since each entry is encoded in at most $(b+1)(\lceil \lg \sigma \rceil + 1) = O(\text{polylog}(n))$ bits, U uses $o(n)$ bits. With U, we can scan A and process b of its elements in constant time; whether or where a boundary element should be created when processing these b elements can be inferred by keeping track of the number of elements that we have scanned so far. Note that at most one boundary element will be created when reading b elements from A, as $b < \lg n < \sigma$. The time needed to create M is hence $O(n'/b) = O(n' \lg \sigma / \lg n)$.

From M we determine the content of $B_0, B_1, \ldots, B_{\sigma-1}$ by constructing a tree T over M similar to large extent to a binary wavelet tree and associating each node u of T with a sequence $M(u)$. At the root node r of T, we set $M(r) = M$, and we perform the following recursive procedure at any node u at level l of T where $l \in [0, \lceil \lg \sigma \rceil - 1]$: We create the left

child, u_0, and the right child, u_1, of u, and perform a linear scan of $M(u)$. During the scan, for each $i \in [0, |M(u) - 1|]$, if $M(u)[i]$ is a boundary element, it is appended to both $M(u_0)$ and $M(u_1)$. If $M(u)[i]$ is not a boundary element and its lth most significant bit is 0, $M(u)[i]$ is appended to $M(u_0)$. If its lth significant bit is 1, it is appended to $M(u_1)$. After generating the sequences $M(u_0)$ and $M(u_1)$, we discard the sequence $M(u)$. We finish recursion after we create $\lceil \lg \sigma \rceil$ levels, i.e., we only examine the first $\lceil \lg \sigma \rceil$ bits of each element of M to determine the tree structure. Thus, this tree has σ leaves, and the sequences associated with the leaves from left to right are named M_0, M_1, ..., $M_{\sigma-1}$. They form a partition of M.

To speed up this process, we use a universal table U'. Recall that $b = \lfloor \frac{\lg n}{2\lceil \lg \sigma \rceil} \rfloor$. U' has an entry for each possible pair (E, c), where E is a sequence of length b drawn from universe $[2\sigma]$ and c is an integer in $[0, \lceil \lg \sigma \rceil - 1]$. This entry stores a pair of packed sequences E_0 and E_1 defined as follows: E_0 or E_1 stores the boundary elements in E and the regular elements in E whose c-th most significant bit is 0 or 1, respectively. The elements in E_0 retain their relative order in E, and the same is true with E_1. As U' has $2^{b \times (\lceil \lg \sigma \rceil + 1)} \times \lceil \lg \sigma \rceil$ entries and each entry stores a pair of packed sequences occupying $O(b\lceil \lg \sigma \rceil)$ bits in total, U' uses $o(n)$ bits. By performing table lookups in U', we can process $M(u)$ in $O(|M(u)| \lg \sigma / \lg n + 1)$ time. Note that we assign n' regular and $2^l \times \frac{n'}{\sigma}$ boundary elements to the nodes at tree level l. Summing over all $O(\sigma)$ nodes of the tree, the total time required to construct this tree is $O(\sum_{l=0}^{\lceil \lg \sigma \rceil - 1}((n' + 2^l \times \frac{n'}{\sigma}) \lg \sigma / \lg n) + \sigma) = O(n' \lg^2 \sigma / \lg n + \sigma)$.

To construct bitvectors B_c for any $0 \le c \le \sigma - 1$, a crucial observation is that the i-th bit in B_c is the same as the least significant bits of the i-th elements of M_c. Thus it takes $O(|B_c|(\lg \sigma + 1)/ \lg n + 1)$ time to compute the content of B_c using bit packing. B_c can then be represented in $O(|B_c|/ \lg n + 1)$ time to support `rank` and `select` by Lemma 5. Summing over all σ bitvectors, the time required to construct B_0, B_1, ..., $B_{\sigma-1}$ from $M_0, M_1, \ldots, M_{\sigma-1}$ is $O(n' \lg \sigma / \lg n + \sigma)$.

Overall, given A, the construction time of these bit vectors is

$$O(n' \lg \sigma / \lg n + (n' \lg^2 \sigma / \lg n + \sigma) + (n' \lg \sigma / \lg n + \sigma)) = O(n' \lg^2 \sigma / \lg n + \sigma). \quad \blacktriangleleft$$

It remains to show how to build all sequences $P_0, P_1, \ldots, P_{\sigma-1}$ efficiently.

▶ **Lemma 11.** *Sequences $P_0, P_1, \ldots, P_{\sigma-1}$ can be constructed in $O(n' \lg^2 \sigma / \lg n + \sigma)$ time.*

Proof. The construction consists of two phases. In the first phase, we compute the set of pairs $R_k = \{(i, \texttt{rank}'(A_k, i)) | 0 \le i \le \sigma - 1\}$ for each chunk A_k. Even though $P_i[k] = \texttt{rank}'(A_k, i)$ and thus the entries of all the P_i's have been computed in this phase, the pairs themselves generated for A_k are not in any order that allows us to directly assign values from these pairs to entries of P_i's quickly enough. Thus, in the second phase, we reorganize all n' pairs computed from all the chunks, to construct $P_0, P_1, \ldots, P_{\sigma-1}$ efficiently.

We first show how to compute the pair set R_k for each A_k efficiently. Let $I[0, \sigma - 1]$ denote a packed sequence such that $I[i] = i$ for each $i \in [0, \sigma - 1]$. Note that I can be constructed once in $O(\sigma)$ time and shared with all chunks. By Lemma 1, a binary wavelet tree, in which node u is associated with $A(u)$ and $I(u)$ as defined before, over A_k could be constructed in $O(\sigma \lg^2 \sigma / \lg n + \sigma)$ time. However, the second term $O(\sigma)$, when summed over all n'/σ chunks, is too expensive to afford. Thus, we modify the structure of a wavelet tree to decrease this term. In the modified tree, when a node v satisfies $|A(v)| \le b = \lfloor \frac{\lg n}{2\lceil \lg \sigma \rceil} \rfloor$, we make v a leaf node without any descendants. With this modification, we observe the following two properties. First, if a leaf node l satisfies $|A(l)| > b$, then the tree level of l must be $\lg \sigma$ and all entries of $A(l)$ store the same symbol. Second, as there are at most $\lceil \sigma / b \rceil$ nodes at each level, the modified tree has $O(\sigma / b \times \lg \sigma) = O(\sigma \lg^2 \sigma / \lg n)$ nodes. The

$O(\sigma)$ term in construction time in Lemma 1 follows from the fact that a wavelet tree has $O(\sigma)$ leaves. With fewer leaves, the modified tree can be constructed in $O(\sigma \lg^2 \sigma / \lg n)$ time. After this tree is constructed, we only keep the sequences $A(l)$ and $I(l)$ for each leaf node l and call them *leaf sequences*. We discard the rest of the tree.

To further compute R_k using these leaf sequences, observe that, for any symbol α, there exists one leaf l such that $A(l)$ contains all the occurrences of α in A. Thus $(I(l)[i], \texttt{rank}'(A_k, I(l)[i])) = (I(l)[i], \texttt{rank}'(A(l), i))$ holds, which we can use to reduce the problem of computing the pairs in R_k to the problem of computing the answer to a partial rank query at each position of $A(l)$ for each leaf l. Hence for each leaf l, we define a packed sequence $Q(l)[0..|A(l)|-1]$ in which $Q(l)[i] = \texttt{rank}'(A(l), i)$ to store these answers. To construct $Q(l)$ efficiently, we consider two cases. When $|A(l)| \leq b$, we apply a universal table U'' to generate $Q(l)$ in constant time. U'' has an entry for each possible pair (F, x), where F is a sequence of length b drawn from universe $[\sigma]$, and x is an integer in $[0, b]$. This entry stores a packed sequence $G[0..x]$ in which $G[i] = \texttt{rank}'(F, i)$. Similar to U in the proof of Lemma 10, U'' uses $o(n)$ bits. When $|A(l)| > b$, all entries of $A(l)$ store the same symbol. Thus, we have $Q(l)[i] = i$ for each $i \in [0, |A(l)|-1]$, and hence we can create $Q(l)$ by copying the first $|A(l)|$ elements from the sequence I which we created before. In either case, $Q(l)$ can be constructed in $O(|A(l)| \lg \sigma / \lg n + 1)$ time. Let l_i denote the $(i+1)$-st leaf visited in a preorder traversal of the tree, and f the number of leaves. Since $\sum_{i=0}^{f} |Q(l_i)| = \sigma$ and $f = O(\sigma \lg^2 \sigma / \lg n)$, the total time required to build $Q(l_0), Q(l_1), \ldots, Q(l_{f-1})$ is $O(\sigma \lg^2 \sigma / \lg n)$. Then we construct the concatenated packed sequence $I_k = I(l_0)I(l_1)\ldots I(l_{f-1})$ and $Q_k = Q(l_0)Q(l_1)\ldots Q(l_{f-1})$. It requires $O(\sigma \lg^2 \sigma / \lg n)$ time to concatenate these sequences if we process $\Theta(\lg n)$ bits, i.e., $O(1)$ words, in constant time by performing bit operations. Since for any $i \in [0, \sigma-1]$, $(I_k[i], Q_k[i])$ is a distinct pair in R_k, I_k and Q_k store all the pairs in R_k. We perform the steps in this and the previous paragraphs for all the chunks in A, and the total time spent in this phase is $O(n' \lg^2 \sigma / \lg n + \sigma)$.

Next we construct $P_0, P_1, \ldots, P_{\sigma-1}$ efficiently using the pairs computed in the previous phase. We first build in $O(n' \lg^2 \sigma / \lg n)$ time two concatenated packed sequences each of length n': $I' = I_0 I_1 \ldots I_{n'/\sigma-1}$ and $Q = Q_0 Q_1 \ldots Q_{n'/\sigma-1}$. Then we construct a binary wavelet tree over I'. Each node, u, of the wavelet tree is associated with two sequences, $I'(v)$ which contains all the elements of I' whose values are within the range represented by v, retaining their relative order in I', and $Q(v)$ in which $Q(v)[i]$ is the element in Q corresponding to $I'(v)[i]$. The wavelet tree construction algorithm of Lemma 1 can be modified easily to construct this wavelet tree in $O(n' \lg^2 \sigma / \lg n + \sigma)$ time. Let l'_i denote the $(i+1)$st leaf of this wavelet tree in preorder. Observe that all the entries in $I'(l'_i)$ store i, and $I'(l'_i)[j]$ initially came from A_j, i.e., $I'(l'_i)[j]$ corresponds to the ith position in chunk A_j. Therefore, $Q(l'_i)[j] = P_i[j]$, and we have $P_i = Q(l'_i)$. The processing time required for this phase is also $O(n' \lg^2 \sigma / \lg n + \sigma)$, which is the same as the bound for the first phase. Therefore, the total time required to construct all sequences $P_0, P_1, \ldots, P_{\sigma-1}$ is $O(n' \lg^2 \sigma / \lg n + \sigma)$. ◀

Combining Lemmas 4, 9, 10 and 11, we have the following result:

▶ **Lemma 12.** *Let $A[0..n'-1]$ be a packed sequence drawn from alphabet $[\sigma]$, where $n' \leq n$ and $\sigma = O(2^{O(\sqrt{\lg n})})$. With the help of a universal table of $o(n)$ bits, a data structure using $n' \lceil \lg \sigma \rceil + o(n' \lg \sigma)$ extra bits can be constructed in $O(n' \lg^2 \sigma / \lg n + \sigma)$ time to support* $\texttt{rank}'$ *queries in $O(1)$ time and $O(1)$ accesses to elements of A.*

4 Fast Construction of Data Structures for Ball Inheritance

We now solve, with fast preprocessing, ball inheritance for the special cases needed later to match the time and space bounds in parts (b) and (c) of Lemma 3. The omitted proofs are deferred to the full version of this paper. One strategy is to construct the solution of Chan et al. [11] by replacing some of their components with those we designed with faster preprocessing. This yields:

▶ **Lemma 13.** *Let $X[0, n-1]$ be a sequence drawn from alphabet $[\sigma]$ denoting the point set $N = \{(X[i], i) | 0 \le i \le n-1\}$, where $2^{\sqrt{\lg n}} \le \sigma \le n$. A $2^{\sqrt{\lg n}}$-ary wavelet tree over X occupying $O(n \lg \sigma \cdot f(\sigma) + n \lg n)$ bits can be constructed in $O(n \lg \sigma / \sqrt{\lg n})$ time to support* `point` *in $O(g(\sigma))$ time and* `noderange` *in $O(\lg \lg n + g(\sigma))$ time, where (a) $f(\sigma) = O(\lg(\lg \sigma / \sqrt{\lg n}))$ and $g(\sigma) = O(\lg(\lg \sigma / \sqrt{\lg n}))$; or (b) $f(\sigma) = O(\lg^{\epsilon} \sigma)$ and $g(\sigma) = O(1)$ for any constant $\epsilon > 0$. The* `noderange` *query requires a universal table of $o(n)$ bits.*

▶ **Lemma 14.** *Let $X[0..n'-1]$ be a packed sequence drawn from alphabet $[\sigma]$ and $Y[0..n'-1]$ be a packed sequence in which $Y[i] = i$ for each $i \in [0..n'-1]$, where $\sigma = O(2^{O(\sqrt{\lg n})})$ and $n' = O(\sigma^{O(1)})$. Given X and Y as input, a d-ary wavelet tree over X using $O(n' \lg \sigma \lg(\lg \sigma / \lg d) + \sigma w)$ bits of space can be constructed in $O(n' \lg^2 \sigma / \lg n + \sigma \log_d \sigma)$ time to support* `point` *in $O(\lg(\lg \sigma / \lg d))$ time and* `noderange` *in $O(\lg \lg \sigma)$ time, where d is a power of 2 upper bounded by $min(\sigma, 2^{\sqrt{\lg n}})$.*

This strategy however cannot achieve, with the preprocessing time as in Lemma 14, part (c) of Lemma 3 when the coordinates of points can be encoded in $O(\sqrt{\lg n})$ bits. For this special case, we twist the approach of Chan et al.: they only store point coordinates explicitly at the leaf level of the wavelet tree, while we take advantage of the smaller grid size to store coordinates at more levels. This allows us to build `rank'` structures at fewer levels of the tree, decreasing the preprocessing time. The details are as follows.

Recall that, when used to represent the given point set N, each node u of the d-ary wavelet tree T is conceptually associated with an ordered list, $N(u)$, of points whose x-coordinates are within the range represented by u, and these points are ordered by y-coordinate. Assume for simplicity that σ is a power of d, and that both $1/\epsilon$ and $\tau = \log_d^{\epsilon} \sigma$ are integers. We assign a color to each level of T: Level 0 is assigned color 0, while any other Level l is assigned color $\max\{c \mid \tau^c \text{ divides } l \text{ and } 0 \le c \le 1/\epsilon - 1\}$. For each node u of T at a level assigned with color $1/\epsilon - 1$, we store the coordinates of the points in $N(u)$ explicitly. For any other node v (let l be the level l of v and c the color assigned to level l), we do not store $N(v)$. Instead, for each $i \in [0, |N(v)|]$, we store a *skipping pointer* $Sp(v)[i]$, which stores, at the closest level l' satisfying $l' > l$ and l' is a multiple of τ^{c+1}, the descendant of v at level l' containing point $N(v)[i]$ in its ordered list of points. This descendant is encoded by its rank among all the descendants of v at level l' in left-to-right order. We use Lemma 12 to support $O(1)$-time `rank'` over $Sp(v)$. Then, since both $N(u)$ and $N(Sp(u)[i])$ order points by y-coordinate, a `rank'`$(Sp(u), i)$ query gives the position of the point $N(u)[i]$ in $N(Sp(u)[i])$. Thus, to compute `point`(v, i), we follow skip pointers starting from v by performing `rank'`, until we reach a level with color $1/\epsilon - 1$, where we retrieve coordinates. With this we have:

▶ **Lemma 15.** *Let $X[0..n'-1]$ be a packed sequence drawn from alphabet $[\sigma]$ and $Y[0..n'-1]$ be a packed sequence in which $Y[i] = i$ for each $i \in [0..n'-1]$, where $\sigma = O(2^{O(\sqrt{\lg n})})$ and $n' = O(\sigma^{O(1)})$. Given X and Y as input, a d-ary wavelet tree over X using $O(n' \lg \sigma \log_d^{\epsilon} \sigma + \sigma w)$ bits for any positive constant ϵ can be constructed in $O(n' \lg^2 \sigma / \lg n + \sigma \log_d \sigma)$ time to support* `point` *in $O(1)$ time and* `noderange` *in $O(\lg \lg \sigma)$ time, where d is a power of 2 upper bounded by $min(\sigma, 2^{\sqrt{\lg n}})$. The* `noderange` *query requires a universal table of $o(n)$ bits.*

5 Optimal Orthogonal Range Reporting with Fast Preprocessing

We now design data structures that support orthogonal range reporting in optimal time and can be constructed fast. Previously, with a solution to ball inheritance, Chan et al. [11] was able to design a relatively simple approach achieving three current best tradeoffs for orthogonal range reporting. However, we have only designed alternative solutions to ball inheritance with fast construction time in special cases. Therefore, we design a different data structure with optimal query time for orthogonal range reporting. The strategy is to use a generalized wavelet tree and our solution to range minimum/maximum (Lemma 8) to reduce the orthogonal range reporting problem in the general case to the special case in which the points are from a $2^{\sqrt{\lg n}} \times n'$ (narrow) grid. In this reduction, we need only support ball-inheritance over a wavelet tree with high fanout which is solved by part (b) of Lemma 13. We further reduce the range reporting problem over points in a narrow grid to this problem over a (small) grid of size at most $2^{\sqrt{\lg n}} \times 2^{2\sqrt{\lg n}}$, to which we can apply Lemma 15 for ball inheritance. Hence we describe our solutions over a small, narrow and general grid in this order, as the solution to the next case uses that to the previous.

5.1 Orthogonal Range Reporting in a Small Grid

▶ **Lemma 16.** *Let N be a set of δ points with distinct y-coordinates in a $2^{\sqrt{\lg n}} \times \delta$ grid where $\delta \leq 2^{2\sqrt{\lg n}}$. Given packed sequences X and Y respectively encoding the x- and y-coordinates of these points where $Y[i] = i$ for any $i \in [0, \delta - 1]$, a data structure occupying $O(\delta \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}})$ bits can be constructed in $O(\delta + \sqrt{\lg n} \cdot 2^{\sqrt{\lg n}})$ time to support orthogonal range reporting over N in $O(\lg \lg n + \mathtt{occ})$ time, where ϵ is an arbitrary positive constant and $\mathtt{occ}$ is the number of reported points.*

Proof. We build a binary wavelet tree T over X augmented with support for ball inheritance. By Lemma 15, T occupies $O(\delta \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}})$ bits and can be built in $O(\delta + \sqrt{\lg n} \cdot 2^{\sqrt{\lg n}})$ time. It also supports `point` in $O(1)$ time and `noderange` in $O(\lg \lg n)$ time. For any internal node v of T, its value array $A(v)$ is built at some point when augmenting T to solve ball inheritance, though $A(v)$ may be discarded eventually. When $A(v)$ was available, we build a data structure $M(v)$ to support range minimum and maximum queries over $A(v)$ using Lemma 8. As T has $\lceil \sqrt{\lg n} \rceil$ non-leaf levels and the total length of the value arrays of the nodes at each tree level is δ, over all internal nodes, these structures use $O(\delta \sqrt{\lg n} \lg \lg n)$ bits in total and the overall construction time is $\sum_v O(|A(v)|/\sqrt{\lg n} + 1) = O(\delta + 2^{\sqrt{\lg n}})$. These costs are subsumed in the storage and construction costs of T. Recall that $A(v)$ stores the x-coordinates of the set, $N(v)$, of points from N whose x-coordinates are within the range represented by v, and the entries of $A(v)$ are ordered by the corresponding y-coordinates of these points. Thus any entry of $A(v)$ can be retrieved by `point` in constant time. Therefore, even after $A(v)$ is discarded, $M(v)$ can still support `rmq`/`rMq` over $A(v)$ in $O(1)$ time.

Given a query range $Q = [a, b] \times [c, d]$, we first locate the lowest common ancestor u of l_a and l_b in constant time, where l_a and l_b denote the a-th and b-th leftmost leaves of T, respectively. Let u_l and u_r denote the left and right children of u, respectively, $[c_l, d_l] = \mathtt{noderange}(c, d, u_l)$ and $[c_r, d_r] = \mathtt{noderange}(c, d, u_r)$. Then $Q \cap N = (([a, +\infty) \times [c_l, d_l]) \cap N(u_l)) \cup (([0, b] \times [c_r, d_r]) \cap N(u_r))$. In this way, we reduce a 2-d 4-sided range reporting in N to 2-d 3-sided range reporting in $N(u_l)$ and $N(u_r)$. To report points in $([a, +\infty) \times [c_l, d_l]) \cap N(u_l)$, we need only report the points in $N(u_l)[c_l, d_l]$ whose x-coordinates are at least a. This can be done by performing range maximum queries over $A(u_l)$ recursively

as follows. We perform $\texttt{rMq}(c_l, d_l)$ to get the index m of the point p that has the maximum x-coordinate in $N(u_l)[c_l, d_l]$, and retrieve its coordinates $(p.x, p.y)$ by $\texttt{point}(u_l, m)$. If $p.x \geq a$, we report p and perform the same process recursively in $N(u_l)[c_l, m-1]$ and $N(u_l)[m+1, d_l]$. Otherwise we stop. The points in $([0, b] \times [c_r, d_r]) \cap N(u_r)]$ can be reported in a similar way. To analyze the query time, observe that we perform `noderange` twice in $O(\lg \lg n)$ time. The recursive procedure is called $O(\texttt{occ})$ times, and each time it is performed, it uses $O(1)$ time. All other steps require $O(1)$ time. Therefore, the overall query time is $O(\lg \lg n + \texttt{occ})$. ◀

5.2 Orthogonal Range Reporting in a Narrow Grid

Our solution for points in a $2^{\sqrt{\lg n}} \times n'$ grid for any $n' \leq n$ uses the following previous result:

▶ **Lemma 17** ([11, Section 2], [6, Lemma 5]). *Given a set, N, of n points in $[u] \times [u]$, a data structure of $O(n \lg^{1+\epsilon} n)$ bits can be constructed in $O(n \lg n)$ time, which supports orthogonal range reporting over N in $O(\lg \lg u + \texttt{occ})$ time, where $\texttt{occ}$ is the number of reported points.*

The following lemma presents our solution for a narrow grid:

▶ **Lemma 18.** *Let N be a set of n' points with distinct y-coordinates in a $2^{\sqrt{\lg n}} \times n'$ grid where $n' \leq n$. Given packed sequences X and Y respectively encoding the x- and y-coordinates of these points where $Y[i] = i$ for any $i \in [0, n'-1]$, a data structure occupying $O(n' \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}} + n'w/2^{\sqrt{\lg n}})$ bits can be constructed in $O(n' + \sqrt{\lg n} \cdot 2^{\sqrt{\lg n}})$ time to support orthogonal range reporting over N in $O(\lg \lg n + \texttt{occ})$ time, where ϵ is an arbitrary positive constant and $\texttt{occ}$ is the number of reported points.*

Proof. Let $b = 2^{2\sqrt{\lg n}}$. We need only consider the case in which $n' > b$ as Lemma 16 applies otherwise. Assume for simplicity that n' is divisible by b. We divide N into n'/b subsets, and for each $i \in [0, n'/b - 1]$, the ith subset, N_i, contains points in N whose y-coordinates are in $[ib, (i+1)b - 1]$. Let p be a point in N_i. We call its coordinates $(p.x, p.y)$ *global coordinates*, while $(p.x', p.y') = (p.x, p.y \bmod b)$ its *local coordinates* in N_i; the conversion between global and local coordinates can be done in constant time. Hence the points in N_i with their local coordinates can be viewed as a point set in a $2^{\sqrt{\lg n}} \times 2^{2\sqrt{\lg n}}$ grid, and we apply Lemma 16 to construct an orthogonal range search structure over N_i.

We also define a point set $\hat{N}$ in a $2^{\sqrt{\lg n}} \times n'/b$ grid. For each set N_i where $i \in [0, n'/b - 1]$ and each $j \in [0, 2^{\sqrt{\lg n}} - 1]$, we store a point (j, i) in $\hat{N}$ iff there exists at least one point in N_i whose x-coordinate is j. Thus the number of points in $\hat{N}$ is at most $n'/b \times 2^{\sqrt{\lg n}} = n'/2^{\sqrt{\lg n}}$. We apply Lemma 17 to construct an orthogonal range search structure over $\hat{N}$. In addition, for each $i \in [0, n'/b - 1]$ and $j \in [0, 2^{\sqrt{\lg n}} - 1]$, we store a list $P_{i,j}$ storing the local y-coordinates of the points in N_i whose x-coordinates are equal to j.

Given a query range $Q = [x_1, x_2] \times [y_1, y_2]$, we first check if $\lfloor y_1/b \rfloor$ is equal to $\lfloor y_2/b \rfloor$. If it is, then the points in the answer to the query reside in the same subset $N_{\lfloor y_1/b \rfloor}$, and we can retrieve these points by performing an orthogonal range query in $N_{\lfloor y_1/b \rfloor}$, which requires $O(\lg \lg n + \texttt{occ})$ time by Lemma 16. Otherwise, we decompose Q into three subranges $Q_1 = [x_1, x_2] \times [y_1, b(\lfloor y_1/b \rfloor + 1) - 1]$, $Q_2 = [x_1, x_2] \times [b(\lfloor y_1/b \rfloor + 1), b\lfloor y_2/b \rfloor - 1]$ and $Q_3 = [x_1, x_2] \times [b\lfloor y_2/b \rfloor, y_2]$. The points in $N \cap Q_1$ and $N \cap Q_3$ are in $N_{\lfloor y_1/b \rfloor}$ and $N_{\lfloor y_2/b \rfloor}$, respectively, and by Lemma 16, they can be reported in $O(\lg \lg n + \texttt{occ}_1)$ and $O(\lg \lg n + \texttt{occ}_3)$ time, respectively, where $\texttt{occ}_1 = |N \cap Q_1|$ and $\texttt{occ}_3 = |N \cap Q_3|$. The points in $N \cap Q_2$ are in $N_{\lfloor y_1/b \rfloor + 1}, N_{\lfloor y_1/b \rfloor + 2}, \ldots, N_{\lfloor y_2/b \rfloor - 1}$. To retrieve them, we first perform an orthogonal range query in $\hat{N}$ with query range $\hat{Q} = [x_1, x_2] \times [\lfloor y_1/b \rfloor + 1, \lfloor y_2/b \rfloor - 1]$. Let (x, y) be a point

in $\hat{N} \cap \hat{Q}$. The existence of this point means that is at least one point in $N_y \cap Q_2$ whose x-coordinates are equal to x; the local y-coordinates of these points are stored in $P_{y,x}$ which we retrieve and convert to global coordinates. After examining all the points in $\hat{N} \cap \hat{Q}$ and retrieving their corresponding points in $N \cap Q_2$ in this way, we have computed all the points in $N \cap Q_2$ in $O(\lg\lg n + \mathtt{occ}_2)$ time where $\mathtt{occ}_2 = |N \cap Q_2|$. The overall query processing time is thus $O(\lg\lg n + \mathtt{occ})$.

To bound the storage costs, by Lemma 16, the orthogonal range reporting structure over each N_i uses $O(2^{2\sqrt{\lg n}} \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}})$ bits. Thus, the range reporting structures over $N_0, N_1, \ldots, N_{n/b-1}$ occupy $O((n'/b) \times (2^{2\sqrt{\lg n}} \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}})) = O(n' \lg^{1/2+\epsilon} n + n'w/2^{\sqrt{\lg n}})$. As there are at most $n'/2^{\sqrt{\lg n}}$ points in $\hat{N}$, by Lemma 17, the range reporting structure for $\hat{N}$ occupies $O(n' \lg^{1+\epsilon} n/2^{\sqrt{\lg n}}) = o(n')$ bits. There are n' points in all $P_{i,j}$'s and each of their local y-coordinates can be encoded in $\lg b = 2\sqrt{\lg n}$ bits. In addition, each $P_{i,j}$ requires a pointer to encode its memory location, so $n'/b \times 2^{\sqrt{\lg n}} = n'/2^{\sqrt{\lg n}}$ pointers are needed. Therefore, the total storage cost of all $P_{i,j}$'s is $O(n'w/2^{\sqrt{\lg n}} + n'\sqrt{\lg n})$. Thus the space costs of all structures add up to $O(n' \lg^{1/2+\epsilon} n + n'w/2^{\sqrt{\lg n}})$ bits. Note that the above analysis assumes $n' > b$. Otherwise, $O(n' \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}})$ bits are needed, so we use $O(n' \lg^{1/2+\epsilon} n + w \cdot 2^{\sqrt{\lg n}} + n'w/2^{\sqrt{\lg n}})$ as the space bound on both cases.

Regarding construction time, when $n' > b$, observe that the point sets $N_0, N_1, \ldots, N_{n'/b-1}$ and $\hat{N}$, as well as the sequences $P[i,j]$ for $i = 0, 1, \ldots, n'/b-1$ and $j = 0, 1, \ldots, 2^{\sqrt{\lg n}} - 1$, can be computed in $O(n')$ time. By Lemma 17, The range reporting structure for $\hat{N}$ can be built in $O(n'/b \times \lg n) = o(n')$ time. Finally, the total construction time of the range reporting structures for $N_0, N_1, \ldots, N_{n/b-1}$ is $O(n'/2^{2\sqrt{\lg n}} \times (2^{2\sqrt{\lg n}} + \sqrt{\lg n} \times 2^{\sqrt{\lg n}})) = O(n')$, which dominates the total preprocessing time of all our data structures. When $n' \leq b$, the construction time is $O(n' + \sqrt{\lg n} \cdot 2^{\sqrt{\lg n}})$ by Lemma 16, so we use $O(n' + \sqrt{\lg n} \cdot 2^{\sqrt{\lg n}})$ as the upper bound on construction time in both cases. ◀

5.3 Orthogonal Range Reporting in an $n \times n$ Grid

We first describe a solution that is slight more general, which requires the grid to be of size $\sigma \times n$ with $2^{\sqrt{\lg n}} \leq \sigma \leq n$, as it will be needed for some applications to be described later.

▶ **Lemma 19.** *Given a sequence $X[0, n-1]$ drawn from alphabet $[\sigma]$ denoting the point set $N = \{(X[i], i) | 0 \leq i \leq n-1\}$, a data structure of $O(n \lg^{1+\epsilon} \sigma + n \lg n)$ bits for any constant $\epsilon > 0$ can be constructed in $O(n \lg \sigma / \sqrt{\lg n})$ time to support orthogonal range reporting over N in $O(\lg\lg n + \mathtt{occ})$ time, where $2^{\sqrt{\lg n}} \leq \sigma \leq n$ and $\mathtt{occ}$ is the number of reported points.*

Proof. We build a $2^{\sqrt{\lg n}}$-ary wavelet tree T upon $X[0, n-1]$ with support for ball inheritance using part (b) of Lemma 13. As in the proof of Lemma 16, for each internal node $v \in T$, we build a data structure $M(v)$ to support range minimum and maximum queries over its value array $A(v)$ in constant time using Lemma 8, even $A(v)$ is not be explicitly stored. Recall that $A(v)$ stores the x-coordinates of the ordered list, $N(v)$, of points from N whose x-coordinates are within the range represented by v, and these points are ordered by y-coordinate. Furthermore, v is associated with another sequence $S(v)$ drawn from alphabet $[2^{\sqrt{\lg n}}]$, in which $S(v)[i]$ encodes the rank of the child of v that contains $N(v)[i]$ in its ordered list. Let $\hat{S}(v)$ denote the point set $\{(S(v)[i], i) | 0 \leq i \leq |S(v)| - 1\}$, and we use Lemma 18 to build a structure supporting orthogonal range reporting over $\hat{S}(v)$.

Given a query range $Q = [a, b] \times [c, d]$, we first locate the lowest common ancestor u of l_a and l_b in constant time, where l_a and l_b denote the a-th and b-th leftmost leaves of T, respectively. Let u_i denote the ith child of u, for any $i \in [0, 2^{\sqrt{\lg n}} - 1]$. We first locate two children, $u_{a'}$ and $u_{b'}$, of u that are ancestors of l_a and l_b, respectively. They can be found in constant time by simple arithmetic as each child of u represents a range of equal size. Then the answer, $Q \cap N$, to the query can be partitioned into three point sets $A_1 = Q \cap N(v_{a'})$, $A_2 = Q \cap (N(v_{a'+1}) \cup N(v_{a'+2}) \cup \ldots N(v_{b'-1}))$ and $A_3 = Q \cap N(v_{b'})$. With $O(\lg \lg n)$-time support for `noderange` and constant-time support for `point` and `rmq`/`rMq`, we can use the algorithm in the proof of Lemma 16 to perform 3-sided range queries over $N(v'_a)$ and $N(v'_b)$ to compute $A_1 \cup A_3$ in $O(\lg \lg n + |A_1| + |A_3|)$ time. To compute A_2, observe that any entry, $\hat{S}(v)[i]$, can be obtained by replacing the x-coordinate of point $N(v)[i]$ with the rank of the child whose ordered list contains $N(v)[i]$. Hence, by performing range reporting over $\hat{S}$ to compute $S \cap ([a' + 1, b' - 1] \times [c_v, d_v])$, where $[c_v, d_v] =$ `noderange`(c, d, v), we can find the set of points in $\hat{S}(v)$ corresponding to the points in A_2. For each point returned, we use `point` to find its original coordinates in N and return it as part of A_2. This process uses $O(\lg \lg n + |A_2|)$ time. Hence we can compute $Q \cap N$ as $A_1 \cup A_2 \cup A_3$ in $O(\lg \lg n +$ `occ`$)$ time.

Now we analyze the space costs. T with support for ball inheritance uses $O(n \lg^{1+\epsilon} \sigma + n \lg n)$ bits for any positive ϵ. For each internal node v, since $w = \Theta(\lg n)$, the data structure for range reporting over $\hat{S}$ uses $O(|S(u)| \lg^{1/2+\epsilon'} n + 2^{\sqrt{\lg n}} \lg n + |S(u)| \lg n / 2^{\sqrt{\lg n}})$ bits for any positive ϵ'. This subsumes the cost of storing $M(u)$ which is $O(|S(u)| \lg \lg n)$ bits. As T has $O(\sigma / 2^{\sqrt{\lg n}})$ internal nodes, the total cost of storing these structures at all internal nodes is $\sum_u O(|S(u)| \lg^{1/2+\epsilon'} n + 2^{\sqrt{\lg n}} \lg n + |S(u)| \lg n / 2^{\sqrt{\lg n}}) = O(n \lg \sigma / \sqrt{\lg n} \times \lg^{1/2+\epsilon'} n + \sigma \lg n) = O(n \lg \sigma \lg^{\epsilon'} n + \sigma \lg n)$. As $\lg n \leq \lg^2 \sigma$ and $\sigma \leq n$, this is bounded by $O(n \lg^{1+2\epsilon'} \sigma)$. Setting $\epsilon' = \epsilon/2$, the space bound turns to be $O(n \lg^{1+\epsilon} \sigma)$ bits. Overall, the data structures occupy $O(n \lg^{1+\epsilon} \sigma + n \lg n)$ bits.

Finally, we analyze the construction time. As shown in Lemma 13, T with support for ball inheritance can be constructed in $O(n \lg \sigma / \sqrt{\lg n})$ time. For each internal node u of T, constructing $M(u)$ and the range reporting structure over $\hat{S}(v)$ requires $O(|S(u)| + \sqrt{\lg n} \cdot 2^{\sqrt{\lg n}})$ time. As T has $O(\sigma / 2^{\sqrt{\lg n}})$ internal nodes, these structures over all internal nodes can be built in $\sum_u O(|S(u)| + \sqrt{\lg n} \times 2^{\sqrt{\lg n}}) = O(n \lg \sigma / \sqrt{\lg n} + \sigma \sqrt{\lg n}) = O(n \lg \sigma / \sqrt{\lg n})$ as $\sigma \leq n$. The preprocessing time of all data structures is hence $O(n \lg \sigma / \sqrt{\lg n})$. ◀

Our result on points over an $n \times n$ gird immediately follows.

▶ **Theorem 20.** *Given a set, N, of n points in rank space, a data structure of $O(n \lg^{1+\epsilon} n)$ bits for any constant $\epsilon > 0$ can be constructed in $O(n\sqrt{\lg n})$ time to support orthogonal range reporting in $O(\lg \lg n +$* `occ`$)$ *time, where* `occ` *is the number of reported points.*

6 Optimal Orthogonal Range Successor with Fast Preprocessing

In this section, we assume that a range successor query asks for the lowest point in the query rectangle. The following theorem presents our result on fast construction of structures for optimal range successor; we provide a proof sketch, while leaving the full proof to the full version of this paper:

▶ **Theorem 21.** *Given n points in rank space, a data structure of $O(n \lg \lg n)$ words can be constructed in $O(n\sqrt{\lg n})$ time to support orthogonal range successor in $O(\lg \lg n)$ time.*

Proof (sketch). Our approach is similar to that in Section 5, but more levels of reductions are required. Let the sequence $X[0, n-1]$ denote the point set $N = \{(X[i], i) | 0 \leq i \leq n-1\}$. We build a $2^{\sqrt{\lg n}}$-ary wavelet tree T upon $X[0, n-1]$ with support for ball inheritance using part (a) of Lemma 13. As shown in the proof of Lemma 19, a query can be answered by locating the lowest common ancestor, u, of the two leaves corresponding to the end points of the query x-range, and then performing two 3-sided queries over the point sets represented by two children of u and one 4-sided query over $S(u)$. For the 3-sided queries, Zhou [32] already designed an indexing structure, which, with our $O(\lg\lg n)$-time support for `point` and `noderange`, can answer a 3-sided query in $O(\lg\lg n)$ time. The construction time is linear, but it is fine since T has only $O(\sqrt{\lg n})$ levels. The 4-side query over $S(u)$ is a range successor query over n' points in a $2^{\sqrt{\lg n}} \times n'$ (medium narrow) grid for any $n' \leq n$.

For such a medium narrow grid, we use the sampling strategy in Lemma 18 to reduce the problem to range successor over a set of n' points in a $2^{\sqrt{\lg n}} \times n'$ grid where $n' \leq 2 \times 2^{2\sqrt{\lg n}} - 1$. The sampling is adjusted, as we need select at most $2^{\sqrt{\lg n}}$ sampled points from each subset. The grid size of $2^{\sqrt{\lg n}} \times n'$ with $n' \leq 2 \times 2^{2\sqrt{\lg n}} - 1$ is the same as that in Lemma 16, so one may be tempted to apply the same strategy of building a binary wavelet tree to reduce it to the problem of building index structures for 3-sided queries. However, we found that, to construct the structure of Zhou [32] over n' points whose coordinates are encoded in $O(\sqrt{\lg n})$ bits, $O(n' \lg\lg n/\sqrt{\lg n})$ time is required, which is a factor of $\lg\lg n$ more than the preprocessing time of the `rmq` structure needed in the proof of Lemma 16. This factor comes from rank reduction in [32], which requires us to sort packed sequences. To overcome this additional cost, we build a $\lg^{1/4} n$-ary wavelet tree over the x-coordinates, whose number of levels is a factor of $O(\lg\lg n)$ less than that of a binary wavelet tree. As discussed for the general case, this strategy reduces the current problem to orthogonal range successor over n' points in an $\lg^{1/4} n \times n'$ (small narrow) grid with $n' \leq n$.

For a small narrow grid, there are two cases. If $n' > \lg n$, we build a binary wavelet tree of height $O(\lg\lg n)$. In the query algorithm, after finding the lowest common ancestor of the two leaves corresponding to the end points of the query x-range, we do not perform 3-sided queries. Instead, we traverse the two paths leading to these two leaves. This requires us to traverse down $O(\lg\lg n)$ levels, and at each level, we perform certain `rank`/`select` operations in constant time, with the right auxiliary structures at each node. No extra support for ball inheritance is needed as we can simply go down the tree level by level to map information. Finally, if $n' < \lg n$, we use sampling to reduce it to even smaller grids of size at most $\lg^{1/4} n \times \lg^{3/4} n$, over which a query can be answered using a table lookup. ◀

7 Applications

We now apply our range search structures to the text indexing problem, in which we preprocess a text string $T \in [\sigma]^n$, where $\sigma \leq n$. Given a pattern string $P[0..p-1]$, a *counting query* computes the number of occurrences of P in T and a *listing query* reports these occurrences.

Text indexing and searching in sublinear time. When both T and P are given in packed form, a text index of Munro et al. [25] occupies $O(n \lg \sigma)$ bits, can be built in $O(n \lg \sigma/\sqrt{\lg n})$ time and supports counting queries in $O(p/\log_\sigma n + \lg n \log_\sigma n)$ time (there are other tradeoffs, but this is their main result). Thus for small alphabet size which is common in practice, they achieve both $o(n)$ construction time and $o(p)$ query time, while previous results achieve at most one of these bounds. To support listing queries, however, they need to increase space cost to $O(n \lg \sigma \lg^\epsilon n)$ bits and construction time to $O(n \lg \sigma \lg^\epsilon n)$, and then a listing query

can be answered in $O(p/\log_\sigma n + \log_\sigma n \lg\lg n + \texttt{occ})$. The increase in storage and construction costs stems from one component they used which is an orthogonal range reporting structure over $t = O(n/r)$ points in a $\sigma^{O(r)} \times t$ grid, for $r = c\log_\sigma n$ for any constant $c < 1/4$. We can apply Lemma 19 over this point set to decrease the construction time of their index for listing queries to match that for counting queries:

▶ **Theorem 22.** *Given a packed text string T of length n over an alphabet of size σ, an index of $O(n \lg\sigma \lg^\epsilon n)$ bits can be built in $O(n \lg\sigma/\sqrt{\lg n})$ time for any positive constant ϵ. Given a packed pattern string P of length p, this index supports listing queries in $O(p/\log_\sigma n + \log_\sigma n \lg\lg n + \texttt{occ})$ time where* occ *is the number of occurrences of P in T.*

Position-restricted substring search. In a position-restricted substring search [23], we are given both a pattern P and two indices $0 \le l \le r \le n-1$, and we report all occurrences of P in $T[l..r]$. Makinen and Navarro [23] solves this problem using an index for the original text indexing problem and a two-dimensional orthogonal range reporting structure. Different text indexes and range reporting structures yield different tradeoffs. The tradeoff with the fastest query time supports position-restricted substring search in $O(p + \lg\lg n + \texttt{occ})$ time, where occ is the output size, and it uses $O(n \lg^{1+\epsilon} n)$ bits and can be constructed in $O(n \lg n)$ time. Again, the construction time of the range reporting structure is the bottleneck, which can be improved by Theorem 20. We can also use a new text index by Bille et al. [7] to achieve speedup when P is given as a packed sequence. We have:

▶ **Theorem 23.** *Given a text T of length n over an alphabet of size σ, an index of $O(n \lg^{1+\epsilon} n)$ bits can be built in $O(n\sqrt{\lg n})$ time for any constant $0 < \epsilon < 1/2$. Given a packed pattern string P of length p, this index supports position-restricted substring search in $O(p/\log_\sigma n + \lg p + \lg\lg\sigma + \texttt{occ})$ time, where* occ *in the size of the output.*

References

1. Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000*, pages 198–207. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892088.
2. Maxim Babenko, Paweł Gawrychowski, Tomasz Kociumaka, and Tatiana Starikovskaya. Wavelet trees meet suffix trees. In *26th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 572–591. Society for Industrial and Applied Mathematics, 2015.
3. Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Linear-time string indexing and analysis in small space. *ACM Transactions on Algorithms (TALG)*, 16(2):1–54, 2020.
4. Djamal Belazzougui and Simon J Puglisi. Range predecessor and lempel-ziv parsing. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2053–2071. Society for Industrial and Applied Mathematics, 2016.
5. Michael A Bender and Martın Farach-Colton. The level ancestor problem simplified. *Theoretical Computer Science*, 321(1):5–12, 2004.
6. Philip Bille and Inge Li Gørtz. Substring range reporting. *Algorithmica*, 69(2):384–396, 2014. doi:10.1007/s00453-012-9733-4.
7. Philip Bille, Inge Li Gørtz, and Frederik Rye Skjoldjensen. Deterministic indexing for packed strings. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, pages 6:1–6:11, 2017. doi:10.4230/LIPIcs.CPM.2017.6.
8. Prosenjit Bose, Meng He, Anil Maheshwari, and Pat Morin. Succinct orthogonal range search structures on a grid with applications to text indexing. In *11th International Symposium on Algorithms and Data Structures*, volume 5664 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2009.

9 Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. In *22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1145, 2011. doi:10.1137/1.9781611973082.85.

10 Timothy M Chan, Meng He, J Ian Munro, and Gelin Zhou. Succinct indices for path minimum, with applications. *Algorithmica*, 78(2):453–491, 2017.

11 Timothy M Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the ram, revisited. In *27th Symposium on Computational Geometry*, pages 1–10. ACM, 2011.

12 Timothy M. Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *21st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010*, pages 161–173, 2010. doi:10.1137/1.9781611973075.15.

13 Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988. doi:10.1137/0217026.

14 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, M. Sohel Rahman, German Tischler, and Tomasz Walen. Improved algorithms for the range next value problem and applications. *Theoretical Computer Science*, 434:23–34, 2012. doi:10.1016/j.tcs.2012.02.015.

15 Maxime Crochemore, Marcin Kubica, Tomasz Walen, Costas S. Iliopoulos, and M. Sohel Rahman. Finding patterns in given intervals. *Fundamenta Informaticae*, 101(3):173–186, 2010. doi:10.3233/FI-2010-283.

16 Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, 40(2):465–492, 2011.

17 Joseph JáJá, Christian Worm Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *15th International Symposium on Algorithms and Computation*, volume 3341 of *Lecture Notes in Computer Science*, pages 558–568. Springer, 2004.

18 Jesper Jansson, Zhaoxian Li, and Wing-Kin Sung. On finding the adams consensus tree. *Information and Computation*, 256:334–347, 2017. doi:10.1016/j.ic.2017.08.002.

19 Marek Karpinski and Yakov Nekrich. Space efficient multi-dimensional range reporting. In *15th Annual International Conference on Computing and Combinatorics (COCOON)*, pages 215–224, 2009. doi:10.1007/978-3-642-02882-3_22.

20 Orgad Keller, Tsvi Kopelowitz, and Moshe Lewenstein. Range non-overlapping indexing and successive list indexing. In *10th Workshop on Algorithms and Data Structures, Proceedings*, volume 4619 of *Lecture Notes in Computer Science*, pages 625–636. Springer, 2007. doi:10.1007/978-3-540-73951-7_54.

21 Hans-Peter Lenhof and Michiel H. M. Smid. Using persistent data structures for adding range restrictions to searching problems. *Informatique Theorique et Applications*, 28(1):25–49, 1994. doi:10.1051/ita/1994280100251.

22 Moshe Lewenstein. Orthogonal range searching for text indexing. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 267–302, 2013. doi:10.1007/978-3-642-40273-9_18.

23 Veli Mäkinen and Gonzalo Navarro. Position-restricted substring searching. In *7th Latin American Symposium on Theoretical Informatics*, pages 703–714. Springer, 2006.

24 Veli Mäkinen and Gonzalo Navarro. Rank and select revisited and extended. *Theoretical Computer Science*, 387(3):332–347, 2007. doi:10.1016/j.tcs.2007.07.013.

25 J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Text indexing and searching in sublinear time. In *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, pages 24:1–24:15, 2020. doi:10.4230/LIPIcs.CPM.2020.24.

26 J Ian Munro, Yakov Nekrich, and Jeffrey S Vitter. Fast construction of wavelet trees. *Theoretical Computer Science*, 638:91–97, 2016.

27 Yakov Nekrich. A data structure for multi-dimensional range reporting. In *23rd ACM Symposium on Computational Geometry (SoCG)*, pages 344–353, 2007. doi:10.1145/1247069.1247130.

28 Yakov Nekrich and Gonzalo Navarro. Sorted range reporting. In *13th Scandinavian Symposium and Workshops, 2012. Proceedings*, pages 271–282, 2012. doi:10.1007/978-3-642-31155-0_24.

29 Mihai Patrascu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *38th Annual ACM Symposium on Theory of Computing, 2006*, pages 232–240. ACM, 2006. doi:10.1145/1132516.1132551.

30 Dan E. Willard. On the application of sheared retrieval to orthogonal range queries. In *2nd Annual ACM SIGACT/SIGGRAPH Symposium on Computational Geometry (SoCG) 1986*, pages 80–89. ACM, 1986. doi:10.1145/10515.10524.

31 Chih-Chiang Yu, Wing-Kai Hon, and Biing-Feng Wang. Improved data structures for the orthogonal range successor problem. *Computational Geometry*, 44(3):148–159, 2011. doi:10.1016/j.comgeo.2010.09.001.

32 Gelin Zhou. Two-dimensional range successor in optimal time and almost linear space. *Information Processing Letters*, 116(2):171–174, 2016. doi:10.1016/j.ipl.2015.09.002.