Optimal Encodings for Range Top-k, Selection, and Min-Max

Paweł Gawrychowski[1] and Patrick K. Nicholson[2(✉)]

[1] Institute of Informatics, University of Warsaw, Warsaw, Poland
[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany
pnichols@mpi-inf.mpg.de

Abstract. We consider encoding problems for range queries on arrays. In these problems the goal is to store a structure capable of recovering the answer to all queries that occupies the information theoretic minimum space possible, to within lower order terms. As input, we are given an array $A[1..n]$, and a fixed parameter $k \in [1, n]$. A *range top-k* query on an arbitrary range $[i, j] \subseteq [1, n]$ asks us to return the ordered set of indices $\{\ell_1, ..., \ell_k\}$ such that $A[\ell_m]$ is the m-th largest element in $A[i..j]$, for $1 \leq m \leq k$. A *range selection* query for an arbitrary range $[i, j] \subseteq [1, n]$ and query parameter $k' \in [1, k]$ asks us to return the index of the k'-th largest element in $A[i..j]$. We completely resolve the space complexity of both of these heavily studied problems—to within lower order terms—for all $k = o(n)$. Previously, the constant factor in the space complexity was known only for $k = 1$. We also resolve the space complexity of another problem, that we call *range min-max*, in which the goal is to return the indices of both the minimum and maximum elements in a range.

1 Introduction

Many important algorithms make use of range queries over arrays of values as subroutines [14,17]. As a prime example, text indexes that support pattern matching queries often maintain an array storing the lengths of the longest common prefixes between consecutive suffixes of the text. During a search for a pattern this array is queried in order to find the position of the minimum value in a given range. That is, a subroutine is needed that can preprocess an array A in order to answer *range minimum queries*. Formally, as input to such a query we are given a range $[i, j] \subseteq [1, n]$, and wish to return the index $k = \arg\min_{i \leq \ell \leq j} A[\ell]$. In text indexing applications memory is often the constraining factor, so the question of how many bits are needed to answer range minimum queries has been heavily studied. After a long line of research (see [2,16]), it has been determined that such queries can be answered in constant time, by storing a data structure of size $2n + o(n)$ bits [7]. Furthermore, this space bound is optimal to within lower order terms (see [7, Sec. 1.1.2]). The interesting thing is that the space does not depend on the number of bits required to store individual

P. Gawrychowski—Currently holding a post-doctoral position at Warsaw Center of Mathematics and Computer Science.

M.M. Halldórsson et al. (Eds.): ICALP 2015, Part I, LNCS 9134, pp. 593–604, 2015.
DOI: 10.1007/978-3-662-47672-7_48

elements of the array A. After constructing the data structure we can discard the array A, while still retaining the ability to answer range minimum queries.

Results of this kind, where it is shown that the solutions to all queries can be stored using less space than is required to store the original array, fall into the category of *encodings*, and, more generally, *succinct* data structures [11]. Specifically, given a set of combinatorial objects χ we wish to represent an arbitrary member of χ using $\lg|\chi| + o(\lg|\chi|)$ bits[1], while still supporting queries, if possible. If queries can be supported by the representation then we refer to it as a data structure, but if not, then we refer to it as an encoding. For the case of range minimum queries or range maximum queries, the set χ turns out to be *Cartesian trees*, which were introduced by Vuillemin [18]. For a given array A, the Cartesian tree encodes the solution to all range minimum queries, and similarly, if two arrays have the same solutions to all range minimum queries, then their Cartesian trees are identical [7].

Recently, there has been a lot of interest the following two problems, that generalize range maximum queries in two different ways. The input to each of the following problems is an array $A[1..n]$, that we wish to preprocess into an encoding occupying as few bits as possible, such that the answers to all queries are still recoverable. We assume a value $k \geq 1$ is fixed at preprocessing time.

- **Range top-k:** Given an arbitrary query range $[i, j] \subseteq [1, n]$ and $k' \in [1, k]$, return the indices of the k' largest values in $[i, j]$. This problem is the natural generalization of range maximum queries and has been the focus of a several papers, leading to asymptotically optimal lower and upper space bounds of $\Omega(n \lg k)$ and $\mathcal{O}(n \lg k)$ bits, proved by Grossi et al. [10] and Navarro, Raman, and Rao [15], respectively. The latter upper bound is a data structure that can answer range top-k' queries in optimal $\mathcal{O}(k')$ time.
- **Range k-selection:** Given an arbitrary query range $[i, j] \subseteq [1, n]$ and $k' \leq k$, return the index of the k'-th largest value in $[i, j]$. This problem was studied in a series of recent papers (see [8] and [3] for further references), culminating in data structures that occupy a linear number of words, and can answer queries in $\mathcal{O}(\lg k' / \lg\lg n + 1)$ time [4]. This query time matches a cell-probe lower bound for near-linear space data structures [12]. It is straightforward to see that any encoding of range top-k queries is also an encoding for range k-selection queries, though the question of how much time is required during a query remains unclear [15]. Very recently, Navarro, Raman, and Rao [15] described a data structure that can be used to answer range k-selection queries in optimal $\mathcal{O}(\lg k' / \lg\lg n + 1)$ time [15], and, like the range top-k data structure, occupies $\mathcal{O}(n \lg k)$ bits of space.

Our Results. We present the first space-optimal encodings to range top-k—and therefore range selection also—as well as a new problem that we call *range min-max*, in which the goal is to return the indices of both the minimum and maximum element in the array. We emphasize that, on their own, the encodings

[1] We use $\lg x$ to denote $\log_2 x$.

Table 1. Old and new results. Both upper and lower bounds are expressed in bits. Our bounds make use of the binary entropy function $H(x) = x\lg(\frac{1}{x}) + (1-x)\lg(\frac{1}{1-x})$. For the entry marked with a † the claimed bound holds when $k = o(n)$.

Ref.	Query	Lower Bound	Upper Bound	Query Time
[7]	max	$2n - \Theta(\lg n)$	$2n + o(n)$	$\mathcal{O}(1)$
[10, 15]	top-k	$\Omega(n \lg k)$	$\mathcal{O}(n \lg k)$	$\mathcal{O}(k')$
[5]	top-2	$2.656n - \Theta(\lg n)$	$3.272n + o(n)$	$\mathcal{O}(1)$
Thm. 1, 2	min-max	$3n - \Theta(\lg(n))$	$3n + o(n)$	$\mathcal{O}(1)$
Thm. 3, 4	top-2	$3nH(\frac{1}{3}) - \Theta(\text{polylog}(n))$	$3nH(\frac{1}{3}) + o(n)$	—
Thm. 3, 4	top-k	$(k+1)nH(\frac{1}{k+1})(1 - o(1))$†	$(k+1)nH(\frac{1}{k+1}) + o(n)$	—

for range top-k and selection do not support queries efficiently: they merely store the solutions to all queries in a compressed form. However, our encoding for range min-max can be augmented with $o(n)$ additional bits of data to create a data structure that supports queries in $\mathcal{O}(1)$ time. Furthermore, even without query support, our encodings for range top-k and selection address a problem posed in the papers of Grossi et al. [10] and Navarro et al. [15].

In Table 1 we present a summary of previous and new results. Prior to this work, the only value for which the exact coefficient of n was known was the case in which $k = 1$ (i.e., range maximum queries). For even $k = 2$ the best previous estimate was that the coefficient of n is between 2.656 and 3.272 [5]. The lower bound of 2.656 was derived using generating functions and an extensive computational search [5]. In contrast, our method is purely combinatorial and gives the exact coefficient for all $k = o(n)$. For $k = 2, 3, 4$ the coefficients are (rounding up) 2.755, 3.245, and 3.610, respectively.

As mentioned above, a negative aspect of our encodings is that they appear to be somewhat difficult to use as the basis for a data structure. However, in the full version [9], we present a data structure based on our encoding that *nearly* matches the optimal space bound. Explicitly, we can achieve a space bound of $(k + 1.5)nH(\frac{1.5}{k+1.5}) + o(n \lg k)$ bits with query time $\mathcal{O}(\text{poly}(k \lg n))$. Thus, our data structure achieves space much closer to the optimal bound than the previous best result [15], but the query time is worse. We leave the following data structure problem open: how can range top-k and selection queries be supported with optimal query time using space matching our encodings (to within lower order terms)?

Finally, we wish to point out that although our formulation of the range top-k problem returns the indices in sorted order, the constant factor in our lower bound also holds for the *unsorted* version, in which we return the indices in an arbitrary order, provided $k = o(n)$. This follows since any encoding strategy for unsorted range top-k can be used to construct a sorted top-k encoding, by padding the end of the input array with $k - 1$ values larger than any other. The unsorted encoding of this padded array can be used to infer the solution to an arbitrary sorted top-k query $[i, j]$ by examining the solutions to queries $[i, j], [i, j + 1], ..., [i, n + k - 1]$: see the full version for details [9].

Discussion of Techniques and Road Map. Prior work for top-k, for $k \geq 2$, focused on encoding a decomposition of the array, called a shallow cutting [10,15]. Since shallow cuttings are a general technique used to solve many other range searching problems [12,13], these previous works [10,15] required additional information beyond storing the shallow cutting in order to recover the answers to top-k queries. Furthermore, in these works the exact constant factor is not disclosed, though we estimate it to be at least twice as large as the bounds we present. For the specific case of range top-2 queries a different encoding has been proposed based on *extended Cartesian trees* [5]. In contrast to both of the previous approaches, our encoding is based the approach of Fischer and Heun [7], who describe what is called a 2D min-heap (resp. max-heap) in order to encode range minimum queries (resp. range maximum queries). We begin in Section 2 by showing how to generalize their technique to simultaneously answer both range minimum and range maximum queries. Our encoding provides the answer to both using $3n + o(n)$ bits in total, compared to $4n + o(n)$ bits using the trivial approach of constructing both encodings separately. We then show this bound is optimal by proving that any encoding for range min-max queries can be used to distinguish a certain class of permutations. We move on in Section 3 to generalize Fischer and Heun's technique in a clean and natural way to larger values of k. Indeed, the encoding we present—like that of Fischer and Heun—is simple enough to implement. The main difficulty is proving that the bound achieved by our technique is optimal. For this we enumerate a particular class of walks, via an application of the so-called cycle lemma of Dvoretzky and Motzkin [6].

Due to lack of space we focus primarily on space lower bounds for encodings. However, in the full version of this paper [9] we show our encoding can be used as the basis for a range top-k data structure. Though the resultant space bound and query time are suboptimal, we note that interesting challenges had to be overcome to design a data structure based on our encoding. Concisely, we required the ability to decompose the encoding into smaller blocks in order to support queries efficiently. To do this we, in some sense, generalized the pioneers approach of Jacobson [11] via a non-trivial decomposition theorem. Since balanced parentheses representations appear in many succinct data structures, we believe this will likely be of independent interest.

2 Optimal Encodings of Range Min-Max Queries

In this section we describe our encoding for range min-max queries. We use $\textsc{RMinMax}(A[i..j])$ to denote a range min-max query on a subarray $A[i..j]$. The solution to the query is the ordered set of indices $\{\ell_1, \ell_2\}$ such that $\ell_1 = \arg\max_{\ell \in [i,j]} A[\ell]$ and $\ell_2 = \arg\min_{\ell \in [i,j]} A[\ell]$.

2.1 Review of Fischer and Heun's Technique

We review the algorithm of Fischer and Heun [7] for constructing the encoding of range minimum (resp. maximum) queries.

Consider an array $A[1..n]$ storing n numbers. Without loss of generality we can alter the values of the numbers so that they are a permutation, breaking ties in favour of the leftmost element. To construct the encoding for range minimum queries we sweep the array from left to right[2], while maintaining a stack. A string of bits $T_{\min}$ (resp. $T_{\max}$) will be emitted in reverse order as we scan the array. Whenever we push an element onto the stack, we emit a one bit, and whenever we pop we emit a zero bit. Initially the stack is empty, so we push the position of the first element we encounter on the stack, in this case, 1. Each time we increment the current position, i, we compare the value of $A[i]$ to that of the element in the position t, that is stored on the top of the stack. While $A[t]$ is not less than (resp. not greater than) $A[i]$, we pop the stack. Once $A[t]$ is less than (resp. greater than) the current element or the stack becomes empty, we push i onto the stack. When we reach the end of the array, we pop all the elements on the stack, emitting a zero bit for each element popped, followed by a one bit.

Fischer and Heun showed that the string of bits output by this process can be used to encode a rooted ordinal tree in terms of its *depth first unary degree sequence* or DFUDS [7]. To extract the tree from a sequence, suppose we read d zero bits until we hit the first one bit. Based on this, we create a node v of degree d, and continue building first child of v recursively. Since there are at most $2n$ stack operations, the tree is therefore represented using $2n$ bits. We omit the technical details of how a query is answered, but the basic idea is to augment this tree representation with succinct data structures supporting navigation operations.

2.2 Upper Bound for Range Min-Max Queries

We propose the following encoding for a simultaneous representation of $T_{\min}$ and $T_{\max}$. Scan the array from left to right and maintain two stacks: a min-stack for range minimum queries, and a max-stack for range maximum queries. Notice that in each step except for the first and last, we are popping an element from exactly one of the two stacks. This crucial observation allows us to save space. We describe our encoding in terms of the min-stack and the max-stack maintained as above. Unlike before however, we maintain two separate bit strings, T and U. If the new element causes $\delta \geq 1$ elements on the min-stack to be popped, then we prepend $0^{\delta-1}1$ to the string T, and prepend 0 to the string U. Otherwise, if the new element causes δ elements on the max-stack to be popped, we prepend $0^{\delta-1}1$ to the string T, and 1 to the string U. Since exactly $2n$ elements are popped during n push operations, the bit string T has length $2n$, and the bit string U has length n, for a total of $3n$ bits.

In the full version [9] we show that by using techniques from succinct data structures it is possible to also support queries on this encoding in $\mathcal{O}(1)$ time.

Theorem 1. *There is a data structure that occupies* $3n + o(n)$ *bits of space, such that any query* $\textsc{RMinMax}(A[i..j])$ *can be answered in* $\mathcal{O}(1)$ *time.*

[2] In the original paper the sweeping process moves from right to left, but either direction yields a correct algorithm by symmetry.

2.3 Lower Bound for Range Min-Max Queries

Given a permutation $\pi = (p_1, ..., p_n)$, we say π contains the permutation pattern s_1-s_2-...-s_m if there exists a subsequence of π whose elements have the same relative ordering as the elements in the pattern. That is, there exist some $x_1 < x_2 < ... < x_m \in [1, n]$ such that for all $i, j \in [1, m]$ we have that $\pi(x_i) < \pi(x_j)$ if and only if $s_i < s_j$. For example, if $\pi = (1, 4, 2, 5, 3)$ then π contains the permutation pattern 1-3-4-2: we use this hyphen notation to emphasize that the indices need not be consecutive. In this case, the series of indices in π matching the pattern are $x_1 = 1$, $x_2 = 2$, $x_3 = 4$ and $x_4 = 5$. If no hyphen is present between elements s_i and s_{i+1} in the permutation pattern, then the indices x_i and x_{i+1} must be consecutive: i.e., $x_{i+1} = x_i + 1$. In terms of the example, π does not contain the permutation pattern 1-34-2.

A permutation $\pi = (p_1, ..., p_n)$ is a *Baxter permutation* if there exist no indices $1 \leq i < j < k \leq n$ such that $\pi(j + 1) < \pi(i) < \pi(k) < \pi(j)$ or $\pi(j) < \pi(k) < \pi(i) < \pi(j+1)$. Thus, Baxter permutations are those that do not contain 2-41-3 and 3-14-2. Permutations with less than 4 elements are trivially Baxter permutations, and for permutations on 4 elements the non-Baxter permutations are exactly $(2, 4, 1, 3)$ and $(3, 1, 4, 2)$. Baxter permutations are well studied, and their asymptotic behaviour is known (see, e.g., OEIS A001181 [1]).

We have the following lemma:

Lemma 1. *Suppose π is a Baxter permutation, stored in an array $A[1..n]$ such that $A[i] = \pi(i)$. If an encoding that can recover all range minimum and maximum queries is constructed on A, then π can be recovered from the encoding.*

Proof. In order to recover the permutation, it suffices to show that we can perform pairwise comparisons on any two elements in A using range minimum and range maximum queries. The proof follows by induction on n.

For the base case, for $n = 1$ there is exactly one permutation, so there is nothing to recover. Thus, let us assume that the lemma holds for all permutations on less than $n \geq 2$ elements. For a permutation on n elements, consider the subpermutation induced by the array prefix $A[1..(n - 1)]$ and suffix $A[2..n]$. These subpermutations must be Baxter permutations, since deleting elements from the prefix or suffix of a Baxter permutation cannot create a 2-41-3 or a 3-14-2. Thus, it suffices to show that we can compare $A[1]$ and $A[n]$, as all the remaining pairwise comparisons can be performed by the induction hypothesis.

Let $x = \text{RMin}(A[1..n])$ and $y = \text{RMax}(A[1..n])$ be the indices of the minimum and maximum elements in the array, respectively. If $x \in \{1, n\}$ or $y \in \{1, n\}$ we can compare $A[1]$ and $A[n]$, so assume $x, y \in [2, n - 1]$. Without loss of generality we consider the case where $x < y$: the opposite case is symmetric (i.e., replacing 3-14-2 with 2-41-3), and $x \neq y$ because $n \geq 2$. Consider an arbitrary index $i \in [x, ..., y]$, and the result of comparing $A[1]$ to $A[i]$ and $A[i]$ to $A[n]$ (that can be done by the induction hypothesis, as $i \in [2, n - 1]$). The result is a partial order on three elements, and is either:

1. One of the two chains $A[1] < A[i] < A[n]$ or $A[n] < A[i] < A[1]$, in which case we are done since $A[1]$ and $A[n]$ can be compared; or

2. A partial order in which $A[i]$ is the minimum or maximum element, and $A[1]$ is incomparable with $A[n]$.

If we are in the latter case for all $i \in [x, y]$, then let $f(i) = 0$ if $A[i]$ is the minimum element in this partial order, and $f(i) = 1$ otherwise. Because of how x and y were chosen, $f(x) = 0$ and $f(y) = 1$. If we consider the values of $f(i)$ for all $i \in [x, y]$, there must exist two indices $i, i+1 \in [x, y]$ such that $f(i) = 0$ and $f(i+1) = 1$. Therefore, the indices $1, i, i+1, n$ form the forbidden pattern 3-14-2, unless $A[1] < A[n]$. □

Theorem 2. *Any data structure encoding range minimum and maximum queries simultaneously must occupy $3n - \Theta(\log n)$ bits, for sufficiently large values of n.*

Proof. Let $L(n)$ be the number of Baxter permutations on n elements. It is known (cf. [1]) that $\lim_{n\to\infty} \frac{L(n)\pi\sqrt{3}n^4}{2^{3n+5}} = 1$. Since we can encode and recover each one by the procedure discussed in Lemma 1, our encoding data structure must occupy at least $\lg L(n) = 3n - \Theta(\log n)$ bits, if n is sufficiently large. □

3 Optimal Encodings for Top-k Queries

In this section we use RTopK($A[i..j]$) to denote a range top-k query on the subarray $A[i..j]$. The solution to such a query is an ordered list of indices $\{\ell_1, ..., \ell_k\}$ such that $A[\ell_m]$ is the m-th largest element in $A[i..j]$.

3.1 Upper Bound for Encoding Top-k Queries

Like the encoding for range min-max queries, our encoding for range top-k queries is based on representing the changes to a certain structure as we scan through the array A. Each prefix in the array will correspond to a different structure. We denote the structure, that we will soon describe, for prefix $A[1..j]$ as $S_k(j)$, for all $1 \le j \le n$. The structure $S_k(j)$ will allow us to answer RTopK($A[i..j]$) for any $i \in [1, j]$. Our encoding will store the differences between $S_k(j)$ and $S_k(j+1)$ for all $j \in [1, n-1]$. Let us begin by defining a single instance for an arbitrary j.

We first define the directed graph $G_j = (V, E)$ with vertices labelled $\{1, ..., j\}$, and where an edge $(i', j') \in E$ iff both $i' < j'$ and $A[i'] < A[j']$ for all $1 \le i' < j' \le j$. We call G_j the *dominance graph* of $A[1..j]$, and say j' *dominates* i', or i' is dominated by j', if $(i', j') \in E$. Next consider the out-degree $d_j(\ell)$ of the vertex labelled $\ell \in [1, j]$ in G_j. We define an array $S[1..j]$, where $S[\ell] = d_j(\ell)$ for $1 \le \ell \le j$. The structure $S_k(j)$ is defined as follows: take the array $S[1..j]$, and for each entry $\ell \in [1, j]$ such that $S[\ell] > k$, replace $S[\ell]$ with k. We use the notation $S_k(j, \ell)$ to refer to the ℓ-th array entry in the structure $S_k(j)$. We refer to an index ℓ to be *active* iff $S_k(j, \ell) < k$, and as *inactive* otherwise. We note that $S_k(n)$ is reminiscent of the one-sided top-k structure of Grossi et al. [10].

Lemma 2. *The total ordering of elements $A[i_1], ..., A[i_{j'}]$, where $\{i_1, ..., i_{j'}\}$ are the active indices in $S_k(j)$, can be recovered by examining only $S_k(j)$.*

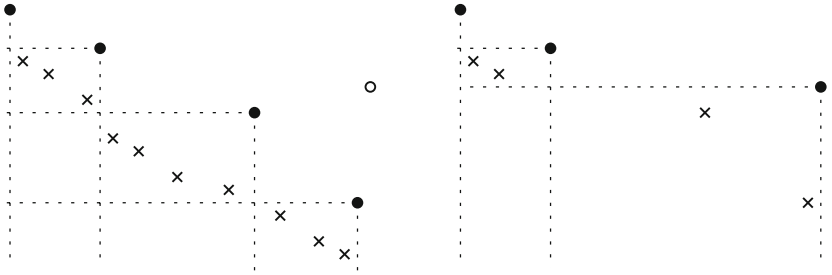

Fig. 1. Geometric interpretation of how the structure $S_k(j)$ is updated to $S_k(j+1)$. In the example $k = 2$, and the value of each active element in the array is represented by its height. Black circles denote 0 values in the array $S_2(j)$, whereas crosses represent 1 values, and 2 values (inactive elements) are not depicted. When the new point (empty circle) is inserted to the structure on the left, it increments the counters of the smallest 10 active elements, resulting in the picture on the right representing $S_2(j+1)$.

Proof. We scan the structure $S_k(j)$ from index j down to 1, maintaining a total ordering on the active elements seen so far. Initially, we have an empty total ordering. At each active location ℓ the value $S_k(j, \ell)$ indicates how many active elements in locations $[\ell+1, j]$ are larger than $A[\ell]$. This follows since an inactive element cannot dominate an active element in the graph G_j. Thus, we can insert $A[\ell]$ into the current total ordering of active elements. □

We define the *size* of $S_k(j)$ as follows: $|S_k(j)| = \sum_{\ell=1}^{j}(k - S_k(j, \ell))$. The key observation is that the structure $S_k(j+1)$ can be constructed from $S_k(j)$ using the following procedure:

1. Compute the value $\delta_j = |S_k(j)| - |S_k(j+1)| + k$. This quantity is always non-negative, as we add one new element to the large staircase, which increases the size by at most k.
2. Find the δ_j indices among the active elements in $S_k(j)$ such that their values in A are the smallest via Lemma 2. Denote this set of indices as $\mathcal{I}$.
3. For each $\ell \in [1, j]$, set $S_k(j+1, \ell) = S_k(j, \ell) + 1$ iff $\ell \in \mathcal{I}$, and $S_k(j+1, \ell) = S_k(j, \ell)$ otherwise.
4. Add the new element at the end of the array, setting $S_k(j+1, j+1) = 0$.

Thus, to construct $S_k(j+1)$ all that is needed is $S_k(j)$ and the value δ_j: see Figure 1. This implies that by storing δ_j for $j \in [1, n-1]$ we can build any $S_k(j)$.

Theorem 3. *Solutions to all queries* $\textsc{RTopK}(A[i..j])$ *can be encoded in at most* $(k+1)nH(\frac{1}{k+1})$ *bits of space.*

Proof. Suppose we store the bitvector $0^{\delta_1}10^{\delta_2}1 \ldots 0^{\delta_{n-1}}1$. This bitvector contains no more than kn zero bits. This follows since each active counter can be incremented k times before it becomes inactive. Thus, storing the bitvector requires no more than $\lg \binom{(k+1)n}{n} \leq (k+1)nH(\frac{1}{k+1})$ bits.

Next we prove that this is all we need to answer a query RTopK$(A[i..j])$. We use the encoding to construct $S_k(j)$. We know that for every element at inactive index ℓ in $S_k(j)$ there are at least k elements with larger value in $A[\ell+1..j]$. Consequently, these elements need not be returned in the solution, and it is enough to recover the indices of the top-k values among the elements at active indices at least i. We apply Lemma 2 on $S_k(j)$ to recover these indices and return them as the solution. □

3.2 Lower Bound for Encoding Top-k Queries

The goal of this section is to show that the encoding from Section 3.1 is, in fact, optimal. The first observation is that all structures $S_k(j)$ for $j \in [1,n]$ can be reconstructed with RTopK queries.

Lemma 3. *Any $S_k(j)$ can be reconstructed with* RTopK *queries.*

Proof. To reconstruct $S_k(j)$, we execute the query RTopK$(A[\ell..j])$ for each $\ell \in [1,j]$. If index ℓ is returned as the k'-th largest element in $[\ell,j]$, then by definition there are exactly $k'-1$ elements in locations $A[\ell+1..j]$ with value larger than $A[\ell]$. Thus, ℓ is an active location and $S_k(j,\ell) = k'-1$. If ℓ is not returned by the query, then it is inactive and we set $S_k(j,\ell) = k$. □

Recall that we encode all structures by specifying $\delta_1, \delta_2, \ldots, \delta_{n-1}$. We call an $(n-1)$-tuple of nonnegative integers $(\delta_1, \delta_2, \ldots, \delta_{n-1})$ *valid* if it encodes some $S_k(1), S_k(2), \ldots, S_k(n)$, i.e., if there exists at least one array $A[1..n]$ consisting of distinct integers such that the structure constructed for $A[1..j]$ is exactly the encoded $S_k(j)$, for every $j = 1, 2, \ldots, n$. Then the number of bits required by the encoding is at least the logarithm of the number of valid $(n-1)$-tuples $(\delta_1, \delta_2, \ldots, \delta_{n-1})$. Our encoding from Section 3.1 shows this number is at most $\binom{(k+1)n}{n}$, but we need to argue in the other direction, which is far more involved.

Recall that the size of a particular $S_k(j)$ is $|S_k(j)| = \sum_{i=1}^{j}(k - S_k(j,i))$. We would like to argue that there are many valid $(n-1)$-tuples $(\delta_1, \delta_2, \ldots, \delta_{n-1})$. This will be proven in a series of transformations.

Lemma 4. *If $(\delta_1, \delta_2, \ldots, \delta_{n-1})$ is valid, then for any $\delta_n \in \{0, 1, \ldots, \lceil \frac{M}{k} \rceil\}$ where $M = \sum_{i=1}^{n-1}(k - \delta_i)$, the tuple $(\delta_1, \delta_2, \ldots, \delta_{n-1}, \delta_n)$ is also valid.*

Proof. Let $A[1..n]$ be an array such that the structure constructed for $A[1..j]$ is exactly $S_k(j)$, for every $j = 1, 2, \ldots, n$. By definition of δ_j, we have that $M = \sum_{i=1}^{n-1}(k - \delta_i) < |S_k(n)|$. Denote the number of active elements in $S_k(j)$ with the corresponding entry set to α as m_α for $\alpha \in [0, k-1]$. For any $s \in \{0, 1, \ldots, \sum_{\alpha=0}^{k-1} m_\alpha\}$, we can adjust $A[n+1]$ so that it is larger than exactly the s smallest active elements in $S_k(n)$. Thus, choosing any $\delta_n \in \{0, 1, \ldots, \sum_{\alpha=1}^{k} m_\alpha\}$ results in a valid $(\delta_1, \delta_2, \ldots, \delta_n)$. Since $|S_k(n)| = \sum_{\alpha=0}^{k-1}(k-\alpha)m_\alpha \le k \sum_{\alpha=0}^{k-1} m_\alpha$, we have $\sum_{\alpha=0}^{k-1} m_\alpha \ge \left\lceil \frac{|S_k(n)|}{k} \right\rceil$, proving the claim. □

Every valid $(n-1)$-tuple $(a_1, a_2, \ldots, a_{n-1})$ corresponds in a natural way to a walk of length $n-1$ in a plane, where we start at $(0,0)$ and perform steps of the form $(1, a_i)$, for $i = 1, 2, \ldots, n-1$. We consider a subset of all such walks. Denoting the current position by (x_i, y_i), we require that a_i is an integer from $[k - \lceil \frac{y_i}{k} \rceil, k]$. Under such conditions, any walk corresponds to a valid $(n-1)$-tuple $(\delta_1, \delta_2, \ldots, \delta_{n-1})$, because we can choose $\delta_i = k - a_i$ and apply Lemma 4. Therefore, we can focus on counting such walks.

The condition $[k - \lceil \frac{y_i}{k} \rceil, k]$ is not easy to work with, though. We will count more restricted walks instead. A Y-restricted nonnegative walk of length n starts at $(0,0)$ and consists of n steps of the form $(1, a_i)$, where $a_i \in Y$ for $i = 1, 2, \ldots, n$, such that the current y-coordinate is always nonnegative. Y is an arbitrary set of integers.

Lemma 5. *The number of valid $(n-1)$-tuples is at least as large as the number of $[k-\Delta, k]$-restricted nonnegative walks of length $n-1-\Delta$.*

Proof. We have already observed that the number of valid $(n-1)$-tuples is at least as large as the number of walks consisting of $n-1$ steps of the form $(1, a_i)$, where $a_i \in [k - \lceil \frac{y_i}{k} \rceil, k]$ for $i = 1, 2, \ldots, n-1$. We distinguish a subset of such walks, where the first Δ steps are of the form $(1, k)$, and then we always stay above (or on) the line $y = k\Delta$. Under such restrictions, $a_i \in [k-\Delta, k]$ implies $a_i \in [k - \lceil \frac{y_i}{k} \rceil, k]$, so counting $[k-\Delta, k]$-restricted nonnegative walks gives us a lower bound on the number of valid $(n-1)$-tuples. □

We move to counting Y-restricted nonnegative walks of length n. Again, counting them directly is non-trivial, so we introduce a notion of Y-restricted returning walk of length n, where we ignore the condition that the current y-coordinate should be always nonnegative, but require the walk ends at $(n, 0)$.

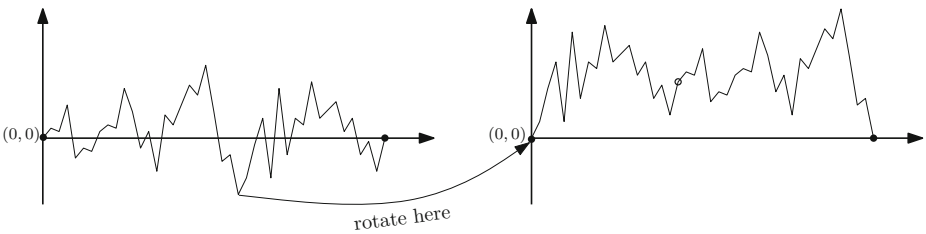

Fig. 2. Left: a Y-restricted walk ending at $(n, 0)$. Right: a cyclic rotation of the walk on the left such that the walk is always nonnegative.

Lemma 6. *The number of Y-restricted nonnegative walks of length n is at least as large as the number of Y-restricted returning walks of length n divided by n.*

Proof. This follows from the so-called cycle lemma [6], but we prefer to provide a simple direct proof. We consider only Y-restricted nonnegative walks of length

n ending at $(n, 0)$, and denote their set by W_1. The set of Y-restricted returning walks of length n is denoted by W_2. The crucial observation is that a cyclic rotation of any walk in W_2 is also a walk in W_2. Moreover, there is always at least one such cyclic rotation which results in the walk becoming nonnegative (see Figure 2). Therefore, we can define a total function $f : W_2 \to W_1$, that takes a walk w and rotates it cyclically as to make it nonnegative. Because there are just n cyclic rotations of a walk of length n, any element of W_1 is the image of at most n elements of W_2 through f. Therefore, $|W_1| \geq \frac{|W_2|}{n}$ as claimed. □

The only remaining step is to count $[k - \Delta, k]$-restricted returning walks of length $n-1-\Delta$. This is equivalent to counting ordered partitions of $k(n-1-\Delta)$ into parts $a_1, a_2, \ldots, a_{n-1-\Delta}$, where $a_i \in [0, \Delta]$ for every $i = 1, 2, \ldots, n-1-\Delta$. This follows since a partition of size ℓ corresponds to a step of size $k - \ell$.

Lemma 7. *The number of ordered partitions of N into g parts, where every part is from $[0, B]$, is at least $\binom{N-2g'+g-1}{g-g'-1}$, where $g' = \lfloor \frac{N}{B} \rfloor$.*

Proof. The number of ordered partitions of N into g parts, where there are no restrictions on the sizes of the parts, is simply $\binom{N+g-1}{g-1}$. To take the restrictions into the account, we first split N into blocks of length B (except for the last block, which might be shorter). This creates $g'+1$ blocks. Then, we additionally split the blocks into smaller parts, which ensures that all parts are from $[0, B]$. We restrict the smaller parts, so that the first and the last smaller part in every block is strictly positive. This ensures that given the resulting partition into parts, we can uniquely reconstruct the blocks. Therefore, we only need to count the number of ways we can split the blocks into such smaller parts, and by standard reasoning this is at least $\binom{N-2g'+g-1}{g-g'-1}$. This follows by conceptually merging the last element in block i with the first element in block $i+1$, so that no further partitioning can happen between them, and then partitioning the remaining set into $g - g'$ pieces. Every such partition corresponds to a distinct restricted partition obtained by splitting between the merged elements, which creates g' additional blocks. □

We are ready to combine all the ingredients. Setting $N = k(n-1-\Delta)$, $g = n-1-\Delta$, $g' = \left\lfloor \frac{k(n-1-\Delta)}{\Delta} \right\rfloor = \left\lfloor \frac{k(n-1)}{\Delta} \right\rfloor - k$ and substituting, the number of bits required by the encoding is:

$$\lg \binom{N-2g'+g-1}{g-g'-1} > \lg \binom{(k+1)(n-2-\Delta-g')}{n-2-\Delta-g'} .$$

Using the entropy function as a lower bound, this is at least $(k+1)n'H(\frac{1}{k+1}) - \Theta(\log n')$, where $n' = n-2-\Delta-g' \geq n(1-\frac{k}{\Delta}) + \frac{k}{\Delta} + k - 2 - \Delta$. Thus, we have the following theorem:

Theorem 4. *For sufficiently large values of n, any data structure that encodes range top-k queries must occupy $(k+1)n'H(\frac{1}{k+1}) - \Theta(\log n')$ bits of space, where $n' \geq n(1-\frac{k}{\Delta}) + \frac{k}{\Delta} + k - 2 - \Delta$, and $\Delta \geq 1$ can be selected to be any positive integer. If $k = o(n)$, then Δ can be chosen such that $\Delta = \omega(k)$ and $\Delta = o(n)$, yielding that the lower bound is $(k+1)nH(\frac{1}{k+1})(1 - o(1))$ bits.*

References

1. OEIS Foundation Inc., The On-Line Encyclopedia of Integer Sequences, Number of Baxter permutations of length n (2011). http://oeis.org/A001181 (Accessed 24 September 2014)
2. Bender, M.A., Farach-Colton, M., Pemmasani, G., Skiena, S., Sumazin, P.: Lowest common ancestors in trees and directed acyclic graphs. Journal of Algorithms **57**(2), 75–94 (2005)
3. Brodal, G.S., Gfeller, B., Jørgensen, A.G., Sanders, P.: Towards optimal range medians. Theoretical Computer Science **412**(24), 2588–2601 (2011)
4. Chan, T.M., Wilkinson, B.T.: Adaptive and approximate orthogonal range counting. In: Proc. of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 241–251. SIAM (2013)
5. Davoodi, P., Navarro, G., Raman, R., Rao, S.: Encoding Range Minima and Range Top-2 Queries. Phil. Trans. R. Soc. A **372**(2016), 1471–2962 (2014)
6. Dvoretzky, A., Motzkin, T.: A problem of arrangements. Duke Mathematical Journal **14**(2), 305–313 (1947)
7. Fischer, J., Heun, V.: Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM J. Comput. **40**(2), 465–492 (2011)
8. Gagie, T., Puglisi, S.J., Turpin, A.: Range quantile queries: another virtue of wavelet trees. In: Karlgren, J., Tarhio, J., Hyyrö, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 1–6. Springer, Heidelberg (2009)
9. Gawrychowski, P., Nicholson, P.K.: Optimal Encodings for Range Min-Max and Top-k. CoRR abs/1411.6581 (2014). http://arxiv.org/abs/1411.6581
10. Grossi, R., Iacono, J., Navarro, G., Raman, R., Rao, S.S.: Encodings for range selection and top-k queries. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 553–564. Springer, Heidelberg (2013)
11. Jacobson, G.: Space-efficient static trees and graphs. In: Proc. of the 30th Annual Symposium on Foundations of Computer Science, pp. 549–554. IEEE (1989)
12. Jørgensen, A.G., Larsen, K.G.: Range selection and median: tight cell probe lower bounds and adaptive data structures. In: Proc. of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 805–813. SIAM (2011)
13. Matoušek, J.: Reporting points in halfspaces. Computational Geometry **2**(3), 169–186 (1992)
14. Navarro, G.: Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. ACM Comput. Surv. **46**(4), 52 (2013)
15. Navarro, G., Raman, R., Satti, S.R.: asymptotically optimal encodings for range selection. In: Proc. 34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS). LIPIcs, vol. 29, pp. 291–301. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2014)
16. Sadakane, K.: Succinct data structures for flexible text retrieval systems. Journal of Discrete Algorithms **5**(1), 12–22 (2007)
17. Skala, M.: Array range queries. In: Brodnik, A., López-Ortiz, A., Raman, V., Viola, A. (eds.) Ianfest-66. LNCS, vol. 8066, pp. 333–350. Springer, Heidelberg (2013)
18. Vuillemin, J.: A unifying look at data structures. Communications of the ACM **23**(4), 229–239 (1980)