On Approximate Range Mode and Range Selection

Hicham El-Zein
Cheriton School of Computer Science, University of Waterloo, Canada
helzein@uwaterloo.ca

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

J. Ian Munro
Cheriton School of Computer Science, University of Waterloo, Canada
imunro@uwaterloo.ca

Yakov Nekrich
Department of Computer Science, Michigan Technological University, USA
yakov@mtu.edu

Bryce Sandlund
Cheriton School of Computer Science, University of Waterloo, Canada
bcsandlund@uwaterloo.ca

Abstract

For any $\varepsilon \in (0, 1)$, a $(1 + \varepsilon)$-approximate range mode query asks for the position of an element whose frequency in the query range is at most a factor $(1 + \varepsilon)$ smaller than the true mode. For this problem, we design a data structure occupying $O(n/\varepsilon)$ bits of space to answer queries in $O(\lg(1/\varepsilon))$ time. This is an encoding data structure which does not require access to the input sequence; the space cost of this structure is asymptotically optimal for constant ε as we also prove a matching lower bound. Furthermore, our solution improves the previous best result of Greve et al. (Cell Probe Lower Bounds and Approximations for Range Mode, ICALP'10) by saving the space cost by a factor of $\lg n$ while achieving the same query time. In dynamic settings, we design an $O(n)$-word data structure that answers queries in $O(\lg n / \lg \lg n)$ time and supports insertions and deletions in $O(\lg n)$ time, for any constant $\varepsilon \in (0, 1)$; the bounds for non-constant $\varepsilon = o(1)$ are also given in the paper. This is the first result on dynamic approximate range mode; it can also be used to obtain the first static data structure for approximate 3-sided range mode queries in two dimensions.

Another problem we consider is approximate range selection. For any $\alpha \in (0, 1/2)$, an α-approximate range selection query asks for the position of an element whose rank in the query range is in $[k - \alpha s, k + \alpha s]$, where k is a rank given by the query and s is the size of the query range. When α is a constant, we design an $O(n)$-bit encoding data structure that can answer queries in constant time and prove this space cost is asymptotically optimal. The previous best result by Krizanc et al. (Range Mode and Range Median Queries on Lists and Trees, Nordic Journal of Computing, 2005) uses $O(n \lg n)$ bits, or $O(n)$ words, to achieve constant approximation for range median only. Thus we not only improve the space cost, but also provide support for any arbitrary k given at query time. We also analyse our solutions for non-constant α.

1   Introduction

The mode and median of a data set are important statistics, widely used across many disciplines. Thus, they are frequently computed in applications for data mining, information retrieval and data analytics. The range mode and median problems further aim at speeding up the computation of the mode and median in an arbitrary subrange of the given sequence of elements, and thus have been studied extensively [17, 1, 19, 20, 12, 13, 16, 2, 15, 4, 11, 5, 14, 7]. In these problems, we preprocess a sequence of elements $c_1, c_2, \ldots, c_n$ to answer queries. Given two indices a and b with $1 \le a \le b \le n$, a *range mode query* asks for a position of the most frequent element in $c_{a..b}$ ($c_{a..b}$ denotes $c_a, \ldots, c_b$), while a *range median query* asks for the position of the median element in $c_{a..b}$. A generalization of range median is the *range selection query*, which asks for the position of the kth smallest element in $c_{a..b}$ for any given k. Thus a range selection query becomes range median if $k = \lceil (b - a + 1)/2 \rceil$.

Due to the massive amounts of electronic data available, linear space data structures are often preferred by modern applications. The following are the best solutions to these query problems that use $O(n)$ words of space. In static settings, Chan et al. [4] showed how to answer a range mode query in $O(\sqrt{n/\lg n})$ time. By proving a conditional lower bound, they also gave strong evidence that, if linear space is required, this query time cannot be improved significantly using purely combinatorial methods with current knowledge. When updates to elements are allowed, El-Zein et al. [7] showed how to support both range mode queries and updates in $O(n^{2/3})$ time. For range selection, the solution of Chan and Wilkinson [5] answers queries in $O(\lg k/\lg \lg n + 1)$ time, matching the lower bound of Jørgensen and Larsen [16] under the cell probe model. He et al. [15] showed how to support range selection in $O((\lg n/\lg \lg n)^2)$ worst-case time and updates in $O((\lg n/\lg \lg n)^2)$ amortized time.

The query times for range mode in both linear space data structure solutions and conditional lower bounds are much larger than that for many other query problems, including range median. To provide faster support for queries, researchers have studied *approximate range mode* [13]. To define this query, let $F_x(c_{a..b})$ denote the frequency of an element x in $c_{a..b}$ and $F(c_{a..b})$ denote the frequency of the mode of $c_{a..b}$ ($F(c_{a..b}) = \max_x F_x(c_{a..b})$). Then a $(1+\varepsilon)$-approximate range mode query asks for the position of an element x in $c_{a..b}$ such that $(1 + \varepsilon) \cdot F_x(c_{a..b}) \ge F(c_{a..b})$ for some positive ε. This element is called a $(1 + \varepsilon)$-approximate mode of $c_{a..b}$. Previously, the best result on this problem is that of Greve et al. [13], which uses $O(n/\varepsilon)$ words of space to support queries in $O(\lg(1/\varepsilon))$ time, for any $\varepsilon \in (0, 1)$.

Approximate range median can be defined similarly. We say that the ith smallest element in the query range $c_{a..b}$ has rank i. Then, for an approximation ratio $\alpha \in (0, 1/2)$, an α-approximate range median query asks for the position of an element x whose rank in $c_{a..b}$ is between $\lceil s/2 \rceil - \alpha s$ and $\lceil s/2 \rceil + \alpha s$, where $s = b - a + 1$. Bose et al. [1] studied this problem, for which they proposed a data structure occupying $O(n/\alpha)$ words of space that answers queries in constant time. An α-approximate range selection query can also be defined, which, for any given k, asks for the position of an element x whose rank in $c_{a..b}$ is between $k - \alpha s$ and $k + \alpha s$. However, this problem has not been formally studied previously.

To further improve the space efficiency of data structures, researchers have recently studied various query problems in the encoding model [8, 14]. Under this model, a data structure is not allowed to store or assume access to the original data set. Instead, it should occupy as little space as possible while providing support for queries. For example, in this model, Fischer and Heun [8] studied the range minimum query problem, which asks for the position of the smallest element in $c_{a..b}$. They proposed a data structure occupying only $2n + o(n)$ bits with constant query time. The range selection problem has also been

Table 1 Static and Dynamic Range Mode Query History. In this table, δ is an arbitrary constant in $(0, 1/2)$ and $m = \min(n \lg n/\varepsilon, n/\varepsilon^2)$.

Query Type	Query Time	Update Time	Space in Bits	Source
Exact	$O(n^\delta \log n)$	-	$O(n^{2-2\delta} \lg n)$	[17]
	$O(1)$	-	$O(n^2 \log \log n/\lg n)$	[20]
	$O(\sqrt{n/\log n})$	-	$O(n \lg n)$	[4]
	$O(n^{3/4} \log n/\log \log n)$	$O(n^{3/4} \log \log n)$	$O(n \lg n)$	[4]
	$O(n^{2/3} \log n/\log \log n)$	$O(n^{2/3} \log n/\log \log n)$	$O(n^{4/3} \lg n)$	[4]
	$O(n^{2/3})$	$O(n^{2/3})$	$O(n \lg n)$	[7]
$(1 + \varepsilon)-$ Approximation	$O(\lg \lg n + \lg(1/\varepsilon))$	-	$O(n \lg n/\varepsilon)$	[1]
	$O(\lg(1/\varepsilon))$	-	$O(n \lg n/\varepsilon)$	[13]
	$O(\lg(1/\varepsilon))$	-	$O(n/\varepsilon)$	new
	$O(\lg m/\lg \lg m)$	$O(\lg n/\varepsilon^2)$	$O(m \lg m)$	new

considered in this model: Grossi et al. [14] proposed an encoding data structure occupying $O(n \lg \kappa)$ bits for any fixed positive integer κ, using which a range selection query can be answered in $O(\lg k/\lg \lg n + 1)$ time for any k given in the query with $1 \le k \le \kappa$.

Naturally, encoding data structures are only relevant when their space occupancy is asymptotically less than the input data, at least for certain choices of parameters. The space costs of previous results on approximate range mode or median, however, match the size of the input sequence asymptotically when ε or α is a constant and become superlinear when ε or α is in $o(1)$. Thus, we study the problem of designing encoding data structures of approximate range mode, median and selection queries, to improve the space efficiency of previous solutions. Furthermore, previously no research has been done on dynamic approximate range mode, while the dynamic exact data structures for range mode require polynomial query and update times. Therefore, we also study approximate range mode queries under dynamic settings, to provide substantially faster support for queries and updates.

Our Results. For $(1 + \varepsilon)$-approximate range mode, where $0 < \varepsilon < 1$, we design an encoding data structure using $O(n/\varepsilon)$ bits that can answer a query in $O(\lg(1/\varepsilon))$ time. This is an improvement upon the previous best result of Greve et al. [13], since we match their query time while saving the space cost by a factor of $\lg n$; we assume a word RAM model in which each word has $\Theta(\lg n)$ bits. We also prove a lower bound to show that any data structure supporting $(1+\varepsilon)$-approximate range mode must use $\Omega(n/(1+\varepsilon))$ bits for any positive ε. This means that our space cost is asymptotically optimal for constant ε. When ε is not necessarily a constant, as long as $\varepsilon = \omega(1/\lg n)$, our data structure uses $o(n \lg n)$ bits, i.e., $o(n)$ words, which is asymptotically less than the space needed to encode the original sequence itself.

For α-approximate range selection, where $0 < \alpha < 1/2$, we design encoding data structures for two variants of this problem. If k is fixed and given in advance, either as a constant or as a function of the size, s, of the query range satisfying certain reasonable constraints (e.g., $k = \lceil s/2 \rceil$ for range median), we have a solution occupying $O(n/\alpha^2)$ bits that can answer a query in constant time. If k is not known beforehand and different values of k could be given with each query, we have another encoding structure in $O(n/\alpha^3)$ bits with constant query time. Our query time matches that of the previous best data structure of Bose et al. [1] which supports range median only, while we decrease the space cost by a factor of $\lg n$ when α is a constant. As we also show that any approximate range selection data structure must use at least $\Omega(n)$ bits, our data structures are asymptotically optimal for constant α.

■ **Table 2** Static and Dynamic Range Median and Selection Query History.

Query Type	Query Time	Update Time	Space in Bits	Source
Exact	$O(1)$	-	$O((n\lg\lg n)^2/\lg n)$	[20]
	$O(\lg n/\lg\lg n)$	-	$O(n\log n)$	[2]
	$O(\lg k/\lg\lg n + 1)$	-	$O(n\lg n)$	[5]
	$O(\lg^2 n)$	$O(\lg^2 n)$	$O(n\lg^2 n)$	[12]
	$O((\lg n/\lg\lg n)^2)$	$O((\lg n/\lg\lg n)^2)$	$O(n\lg^2 n/\lg\lg n)$	[2]
	$O((\lg n/\lg\lg n)^2)$	$O((\lg n/\lg\lg n)^2)$	$O(n\lg n)$	[15]
α – Approximation (with fixed k)	$O(1)$	-	$O(n\lg n/\alpha)$	[1]
	$O(1)$	-	$O(n/\alpha^2)$	new
α – Approximation	$O(1)$	-	$O(n/\alpha^3)$	new

In dynamic settings, for any $\varepsilon \in (0, 1)$, we present an $O(m\lg m)$-bit structure where $m = \min(n\lg n/\varepsilon, n/\varepsilon^2)$. It supports $(1+\varepsilon)$-approximate range mode in $O(\lg m/\lg\lg m)$ time and insertions/deletions in $O(\lg n/\varepsilon^2)$ time. When ε is an arbitrary constant in $(0, 1)$, this data structure uses $O(n)$ words, answers queries in $O(\lg n/\lg\lg n)$ time, and supports updates in $O(\lg n)$ time. As the best result on dynamic exact range mode [7] requires $O(n^{2/3})$ time for both queries and updates, this approximate solution is much faster for constant ε. It is also the first result on dynamic approximate range mode. Finally, we apply the technique to solve static $(1+\varepsilon)$-approximate three-sided range mode in two dimensions, achieving $O(\lg m)$ time query and occupying $O(m\lg m)$ words of space, where again $m = \min(n\lg n/\varepsilon, n/\varepsilon^2)$. This is another new approximate query problem.

Tables 1 and 2 compare our results to previous work to be surveyed in Section 2.

2   Previous Work

Range Mode. Krizanc et al. [17] first studied the static range mode problem and showed that, for any $\delta \in (0, 1/2)$, there is an $O(n^{2-2\delta})$-word solution that answers queries in $O(n^\delta \log n)$ time. Setting $\delta = 1/2$ yields an $O(n)$-word data structure supporting range mode in $O(\sqrt{n}\log n)$ time. They also presented a data structure using $O(n^2\log\log n/\log n)$ words, or $O(n^2\log\log n)$ bits, to support queries in constant time. Chan et al. [4] further provided a better linear word solution with $O(\sqrt{n/\log n})$ query time. They also proved a conditional lower bound to show that, with current knowledge, either the query time must be polynomial, or the construction time must be polynomially larger than n. Later, Greve et al. [13] gave an (unconditional) lower bound in the cell probe model, showing that any structure using S memory cells of w-bit words requires $\Omega(\frac{\log n}{\log(Sw/n)})$ time to answer a range mode query. On the other end of the spectrum, there has been work [19, 20] on improving the constant-time query structure of Krizanc et al., and the best solution uses $O(n^2\lg\lg n/\lg^2 n)$ words, or $O(n^2\lg\lg n/\lg n)$ bits [20].

In dynamic settings, Chan et al. [4] provided a tradeoff among space cost, query time and update time. This tradeoff implies two important results: using linear space in words, range mode can be supported in $O(n^{3/4}\log n/\log\log n)$ worst-case time while updates can be performed in $O(n^{3/4}\log\log n)$ amortized expected time. Alternatively, they can use $O(n^{4/3})$ words to improve the query and update efficiency to $O(n^{2/3}\log n/\log\log n)$ worst-case time and amortized expected time, respectively. They also proved a conditional lower bound to show that, with current knowledge, either queries or updates must require polynomial time. Very recently, El-Zein et al. [7] further improved these solutions by designing an $O(n)$-word structure supporting both queries and updates in $O(n^{2/3})$ time.

Bose et al. [1] were the first to study approximate range mode. They showed how to provide constant-time support for 4-approximate mode, 3-approximate mode and 2-approximate mode using data structures occupying $O(n)$, $O(n \lg \lg n)$ and $O(n \lg n)$ words, respectively. For $(1 + \varepsilon)$-approximation, they designed an $O(n/\varepsilon)$-word solution that can answer a query in $O(\lg \lg_{1+\varepsilon} n) = O(\lg \lg n + \lg(1/\varepsilon))$ time. Greve et al. [13] further improved these results by using $O(n/\varepsilon)$ words of space to support queries in $O(\lg(1/\varepsilon))$ time.

Range Median and Selection. The study of range median also has a rich history. It was also Krizanc et al. [17] who initially proposed this problem. There have been several solutions with near-quadratic space and constant query time [17, 19, 20], the best of which uses $O((n \lg \lg n / \lg n)^2)$ words [20]. For linear-space solutions, following a series of earlier work [17, 10, 12, 3], Brodal et al. [2] first achieved an $O(n)$-word solution that answers range median and selection queries in $O(\lg n / \lg \lg n)$ time. Jørgensen and Larsen [16] further improved the query time of range selection to $O(\lg \lg n + \lg k / \lg \lg n)$, where k is the specified query rank. They also proved that, under the cell probe model, $\Omega(\lg k / \lg \lg n + 1)$ time is necessary for any range selection data structure using $O(n \lg^{O(1)} n)$ space. Chan and Wilkinson [5] were then the first who designed a linear word solution with $O(\lg k / \lg \lg n + 1)$ optimal query time for range selection. More recently, Grossi et al. [14] proposed an encoding data structure occupying $O(n \lg \kappa)$ bits for any fixed positive integer κ, using which a range selection query can be answered in $O(\lg k / \lg \lg n + 1)$ time for any k given in the query with $1 \le k \le \kappa$. Gawrychowski and Nicholson [11] presented a space-optimal encoding of range selection which uses even less space, and proved its space cost is optimal within an $o(n)$ additive term in bits, though no support for queries is provided. All of the above results for range selection assume the selection rank k is specified at query time.

In the dynamic case, Gfeller and Sanders [12] proposed a data structure that uses $O(n \lg n)$ words of space to support range median in $O(\lg^2 n)$ time and insertions and deletions in $O(\lg^2 n)$ amortized time. The structure of Brodal et al. [2] occupies $O(n \lg n / \lg \lg n)$ words of space, answers queries in $O((\lg n / \lg \lg n)^2)$ worst-case time and supports insertions and deletions in $O((\lg n / \lg \lg n)^2)$ amortized time. Later He et al. [15] improved the space cost to $O(n)$ words while providing the same support for queries and updates. The work of Bose et al. [1] is the only work on α-approximate range median, for which they proposed a data structure occupying $O(n/\alpha)$ words of space that answers queries in constant time.

3   Approximate Range Mode

Before we proceed, we give a few preliminaries. We will at times refer to elements (of $c_{1..n}$ or otherwise) as *colors*. This is because their data type has no significance in frequency applications and thus the term color standardizes the data type. Furthermore, at times we create indexing such as a value r_i for when the mode in some range $c_{s_i..r_i}$ exceeds a given threshold. It is possible the mode never exceeds such a threshold. To avoid dealing with such corner cases in the rest of this exposition, we make the assumption that our list of elements $c_{1..n}$ is padded at the beginning and end with a sufficient number of one arbitrary color.

We allow non-constant ε. However, in our upper bounds, we make the restriction $\epsilon \le 1$, to allow simplification in the runtime and space analyses.

▶ **Theorem 1.** *Any one-dimensional $(1 + \varepsilon)$-approximate range mode data structure requires $\Omega(n/(1 + \varepsilon))$ bits.*

Proof. Using a simple proof we show that $\Omega(n/(1+\varepsilon))$ bits are required for any data structure that answers one-dimensional approximate range mode queries. Here we allow arbitrary ϵ.

Given an approximation factor $1 + \varepsilon$, divide the sequence S of size n into $\lfloor n/(2k) \rfloor$ *full blocks* each of size $2k$, where $k = \lceil 1 + \varepsilon \rceil + 1$, and, if n is not a multiple of $2k$, a non-full block of size $n \bmod 2k$. Denote by $t_1, \ldots, t_{k+1}$ $k + 1$ arbitrary, distinct colors. We say that S satisfies property $(*)$ if for each full block b in S one of the following two conditions hold:

- either b consists of t_1 repeated k times followed by $t_2, \ldots, t_{k+1}$,
- or b consists of $t_2, \ldots, t_{k+1}$ followed by t_1 repeated k times.

Clearly, the number of sequences that satisfy $(*)$ is at least $2^{\lfloor n/(2k) \rfloor}$, since there exist $\lfloor n/(2k) \rfloor$ full blocks in a sequence of size n and each of them can have one of two different values. Moreover, for any two distinct sequences S_1 and S_2 satisfying $(*)$ differing at full block b, there exists at least one approximate range mode query, namely the query that asks for an approximate mode of b, that will return different values (either a value from the first k position in the block or from the last k positions of the block). Thus, the information theoretic lower bound for storing an approximate range mode data structure is $\Omega(\lg 2^{\lfloor n/(2k) \rfloor}) = \Omega(\lfloor n/(2k) \rfloor) = \Omega(n)$ bits.   ◀

We now proceed with our new upper bound. Our data structure consists of two parts. The first part answers *low* frequency queries $c_{a..b}$ with $F(c_{a..b}) \leq \lceil 1/\varepsilon \rceil$, and is exact. The second part answers *high* frequency queries $c_{a..b}$ with $F(c_{a..b}) > \lceil 1/\varepsilon \rceil$, and makes use of the approximation factor.

Low Frequencies: $O(n/\varepsilon)$-Bits $O(\lg(1/\varepsilon))$ Query Time. Similar to the data structure of Greve et al. [13], for $k = 0, \ldots, \lceil 1/\varepsilon \rceil$ let Q_k be an increasing sequence of size n such that $Q_k[i]$ is the largest integer $j \geq i$ satisfying $F(c_{i..j}) = k$. Since Q_k is an increasing sequence whose largest element is n, we store it in $2n + O(n/\lg^2 n)$ bits [18] while still accessing its ith element in constant time[1]. The total space used is $O(n/\varepsilon)$ bits. Given a query range $c_{a..b}$, $F(c_{a..b}) > k$ iff $b > Q_k[a]$. Thus, using binary search, we can determine if $F(c_{a..b}) < 1/\varepsilon$ and $K = F(c_{a..b})$ in that case. If $F(c_{a..b}) < 1/\varepsilon$ we return index $Q_{(K-1)}[a] + 1$; otherwise we query the high frequency structure. The total time is $O(\lg(1/\varepsilon))$.

High Frequencies: $O(n/\varepsilon)$-Bits $O(\lg \lg n + \lg(1/\varepsilon))$ Query Time. We first present an $O(n/\varepsilon)$-bit structure that answers high frequency $(1 + \varepsilon)$-approximate range mode queries in $O(\lg \lg n + \lg(1/\varepsilon))$ time. We start by developing a tool to binary search the frequency of the mode, with the goal of locating a $(1 + \varepsilon)$-approximate mode.

▶ **Lemma 2.** *There exists a data structure using $O(k \cdot \varepsilon \cdot n/(1 + \varepsilon)^k + n/\lg^2 n)$ bits that can find in constant time, for any query range $c_{a..b}$, one of the following that holds:*
1. $F(c_{a..b}) < (1 + \varepsilon)^k/\varepsilon$,
2. $F(c_{a..b}) > (1 + \varepsilon)^k/\varepsilon$, *or*
3. $((1 + \varepsilon)^{k-1/2})/\varepsilon < F(c_{a..b}) < ((1 + \varepsilon)^{k+1/2})/\varepsilon$.
In case 2, we find an element with frequency greater than $(1 + \varepsilon)^k/\varepsilon$ in range $c_{a..b}$. In case 3, we find an element with frequency greater than $((1 + \varepsilon)^{k-1/2})/\varepsilon$ in range $c_{a..b}$.

When this structure is present for all k in range $0, \ldots, \lfloor \lg_{1+\varepsilon} \varepsilon n \rfloor$, the above trichotomy is sufficient to binary search for an approximate mode of frequency at least $1/\varepsilon$. If we ever land in case 3, the encoding gives an approximate mode, and otherwise, we find the k satisfying

---

[1] We store $Q_k[1]$ and $(Q_k[i] - Q_k[i-1])$ in unary with a 0 separator between each two consecutive values in a $2n$-bit vector ψ with rank and select structures. To access $Q_k[i]$ we count the number of 1s before the ith 0 in ψ.

$(1 + \varepsilon)^k / \varepsilon < F(c_{a..b}) < (1 + \varepsilon)^{k+1} / \varepsilon$, which represents case 2 for value k and case 1 for value $k + 1$. Since case 2 provides an element with frequency greater than $(1 + \varepsilon)^k / \varepsilon$, this element is an approximate mode.

Proof. Let $1 + \Delta = \sqrt{1 + \varepsilon}$ and $f_j = (\Delta/\varepsilon) \cdot (1 + \Delta)^j$. For each integer i in $[0, n/\lceil f_{2k-1}\rceil]$ let $s_i = i \cdot \lceil f_{2k-1} \rceil + 1$ and denote by r_i the smallest value such that $F(c_{s_i..r_i}) \geq (1 + \Delta)^{2k}/\varepsilon$. Notice that c_{r_i} is the unique mode of $c_{s_i..r_i}$. Similarly, for each integer i in $[0, n/\lceil f_{2k} \rceil]$, let $s'_i = i \cdot \lceil f_{2k} \rceil + 1$ and denote by r'_i the smallest value such that $F(c_{s'_i..r'_i}) \geq (1 + \Delta)^{2k+1}/\varepsilon$.

Given a query range $c_{a..b}$, we find the biggest indices s_i, s'_j preceding or equal to a. We proceed as follows.

1. If $b < r_i$, then $F(c_{a..b}) \leq F(c_{s_i..r_i-1}) < ((1 + \Delta)^{2k}/\varepsilon) = ((1 + \varepsilon)^k/\varepsilon)$.
2. If $b \geq r'_j$, then $F_{r'_j}(c_{a..b}) > F_{r'_j}(c_{s'_j..r'_j}) - f_{2k}$, since there are at most $\lceil f_{2k} \rceil - 1 < f_{2k}$ elements between s'_j and a. Then:

$$F_{r'_j}(c_{a..b}) > F_{r'_j}(c_{s'_j..r'_j}) - f_{2k} \geq ((1 + \Delta)^{2k+1}/\varepsilon) - (\Delta/\varepsilon) \cdot (1 + \Delta)^{2k} = (1 + \Delta)^{2k}/\varepsilon$$
$$= (1 + \varepsilon)^k/\varepsilon.$$

3. Suppose $b \geq r_i$ and $b < r'_j$. Since there are at most $\lceil f_{2k-1} \rceil - 1 < f_{2k-1}$ elements between s_i and a and since $b \geq r_i$, we have that

$$F_{r_i}(c_{a..b}) > F_{r_i}(c_{s_i..r_i}) - f_{2k-1} \geq ((1 + \Delta)^{2k}/\varepsilon) - (\Delta/\varepsilon) \cdot (1 + \Delta)^{2k-1} = ((1 + \Delta)^{2k-1})/\varepsilon$$
$$= ((1 + \varepsilon)^{k-1/2})/\varepsilon.$$

Finally, since $b < r'_j$, then $F(c_{a..b}) \leq F(c_{s'_j..r'_j-1}) < ((1 + \Delta)^{2k+1})/\varepsilon = ((1 + \varepsilon)^{k+1/2})/\varepsilon$.

To store the values $\{r_i\}$, we construct a bit vector of length $O(n)$ as follows. In the bit vector, there are n 0s. For each r_i, we insert a 1 bit after the r_ith 0 bit in the bit vector. Thus r_i is equal to the number of 0s before the ith 1 bit in the bit vector. A second bit vector of length $O(n)$ is used to encode the values $\{r'_i\}$ in a similar way. We then represent these two bit vectors in the succinct data structure of Patrascu [18]. This data structure provides constant time rank and select, which allow us to locate r_i and r'_j, and thus determine whether case 1, 2, or 3 applies, in constant time.

For a bit vector of size n with m ones, the space cost can be made $O(m \lg(n/m) + n/\lg^2 n)$ bits [18]. For vector r, $m \lg(n/m) = O((n/f_{2k-1}) \lg f_{2k-1})$, and for vector r', $m \lg(n/m) = O((n/f_{2k}) \lg f_{2k})$. The cost is dominated by vector r. Let us first consider the $O(m \lg(n/m))$ term. We have

$$n/f_{2k-1} \lg f_{2k-1} = \frac{\varepsilon n}{\Delta (1 + \Delta)^{2k-1}} \lg((\Delta/\varepsilon) \cdot (1 + \Delta)^{2k-1}). \tag{1}$$

Rationalizing the denominator, we can show $\frac{1}{\Delta} = \frac{1}{\sqrt{1+\varepsilon}-1} = \frac{1+\sqrt{1+\varepsilon}}{\varepsilon}$ and so $\frac{1}{\Delta} = \Theta(\frac{1}{\varepsilon})$ and $\Delta/\varepsilon = O(1)$. Thus, with $\epsilon \leq 1$, we can bound (1) with $O\left(\frac{(k-1/2)n}{(1+\varepsilon)^{k-1/2}} \lg(1 + \varepsilon)\right)$. Finally, since we restrict $\varepsilon \leq 1$, we can do a Taylor series expansion to give $\lg(1 + \varepsilon) = O(\varepsilon)$. Thus our final space bound is $O((n/f_{2k-1}) \lg f_{2k-1} + n/\lg^2 n) = O(k \cdot \varepsilon \cdot n/(1 + \varepsilon)^k + n/\lg^2 n)$. ◀

To make the above lemma useful, we must apply it to all k in range $0, \ldots, \lfloor \lg_{1+\varepsilon} \varepsilon n \rfloor$. We first analyze the total space cost of all the $O(k \cdot \varepsilon \cdot n/(1+\varepsilon)^k)$ terms. Summing up these terms, we have $O\left(\sum_{k=1}^{\lfloor \lg_{1+\varepsilon} n \rfloor}(k \cdot \varepsilon \cdot n/(1+\varepsilon)^k)\right) = O\left(n \cdot \varepsilon \sum_{k=1}^{\infty}(k/(1+\varepsilon)^k)\right) = O(n/\varepsilon)$ bits. The other term comes out to $O(\lg_{1+\varepsilon}(\varepsilon \cdot n) \cdot n/\lg^2 n) \subseteq O\left(\frac{n}{\lg n \lg(1+\varepsilon)}\right)$ bits. Again applying Taylor series for $1/\lg(1 + \varepsilon) = O(1/\varepsilon)$ gives $O(n/(\varepsilon \lg n))$ bits. Thus the total space cost is $O(n/\varepsilon)$ bits.

The time complexity of the binary search is different from a typical binary search. The number of values of k in the entire range is $O(\lg_{1+\varepsilon} n)$, so the complexity of the binary search is $O(\lg(\lg_{1+\varepsilon} n)) = O(\lg\left(\frac{\lg n}{\lg(1+\varepsilon)}\right)) = O(\lg\left(\frac{\lg n}{\varepsilon}\right)) = O(\lg\lg n + \lg(1/\varepsilon))$.

▶ **Lemma 3.** *There exists an $O(n/\varepsilon)$-bit data structure that supports one-dimensional $(1+\varepsilon)$-approximate range mode queries in $O(\lg\lg n + \lg(1/\varepsilon))$ time.*

High Frequencies: $O(n/\varepsilon)$-Bits $O(\lg(1/\varepsilon))$ Query Time.   The bottleneck of the approach described in the previous section is the binary search on k. To speed up queries, we store an additional data structure that uses $O(n)$ bits but returns a 4-approximate range mode.

▶ **Lemma 4.** *There exists an $O(n)$-bit data structure that supports one-dimensional approximate range mode queries in constant time with approximation factor 4.*

Proof. We assume n is a power of 2. We construct a network of fusion trees [9]. At the top level, we store two fusion trees $\mathcal{F}_{n/2,l}$ and $\mathcal{F}_{n/2,r}$. The tree $\mathcal{F}_{n/2,l}$ contains the values $e_1, \ldots, e_{\lg n}$, where e_j is the largest index satisfying $F(c_{e_j..n/2}) = 2^j$. $\mathcal{F}_{n/2,r}$ contains the values $e_1, \ldots, e_{\lg n}$, where e_j is the smallest index satisfying $F(c_{n/2..e_j}) = 2^j$. If a query crosses the middle index $n/2$, we query $\mathcal{F}_{n/2,l}$ to get p_1, the smallest value greater than or equal to a, and we query $\mathcal{F}_{k,r}$ to get p_2, the largest value less than or equal to b. We return p_1 if $F(c_{p_1..n/2}) > F(c_{n/2..p_2})$ and p_2 otherwise. Clearly, p_1 is a 2-approximate mode for $c_{a..n/2}$ and p_2 is a 2-approximate mode for $c_{n/2..b}$. The true mode has at least half its occurrences in one of these regions, so the value we return is a 4-approximate mode for $c_{a..b}$.

If the query does not cross the middle, it falls entirely in one of the two sides. We may therefore repeat our fusion tree scheme in a divide and conquer fashion, recursing on the two halves. Eventually, there will be a level of the recursion that intersects the query.

To analyze the total space used, we use the recurrence $S(n) = 2S(n/2) + O(\lg^2 n)$, which solves to $S(n) = O(n)$ bits.

To analyze the time complexity of the query, observe that the fusion trees on $O(\lg n)$ elements with word size $O(\lg n)$ support the necessary predecessor/ successor queries in constant time. However, we must know which fusion trees to query. This involves finding the level of recursion in which the query range intersects the midpoint. This is equivalent to the highest set bit in the XOR of a and b, which can be determined in constant time in the word RAM model. With this information, we can do the necessary arithmetic to find the appropriate fusion trees to query, and thus query takes constant time.   ◀

We now return to the $(1+\varepsilon)$-approximation. To answer a query $c_{a..b}$, we first query the 4-approximation structure of Lemma 4, which returns a corresponding frequency x. We now know $x \le F(c_a, \ldots, c_b) \le 4x$. We have thus shrunk the number of values of k from Lemma 2 that need be tested for the $(1+\varepsilon)$-approximation from $\lceil \lg_{1+\varepsilon} n \rceil$ to $\lceil \lg_{1+\varepsilon}(4x/x) \rceil = \lceil \lg_{1+\varepsilon} 4 \rceil$. Thus our binary search now takes time $O(\lg\left(\frac{2}{\lg(1+\varepsilon)}\right)) = O(\lg(1/\varepsilon))$.

▶ **Theorem 5.** *There exists an $O(n/\varepsilon)$-bit data structure that supports one-dimensional approximate range mode queries in $O(\lg(1/\varepsilon))$ time with approximation factor $1+\varepsilon$.*

4   Dynamic Approximate Range Mode

In this section we consider the dynamic variant of the approximate range mode problem. We maintain our sequence $c_{a..b}$ under insertions and deletions, so that for an arbitrary query range $c_{a..b}$ an approximate range mode can be found efficiently.

The high-level approach is as follows. Similar to Section 3, for each $j \leq \lg_{1+\varepsilon} n$, our goal is to maintain a set of intervals $\mathcal{I}_j$ such that the mode of a query range $c_{a..b}$ occurs more than $(1+\varepsilon)^j$ times if and only if $c_{a..b}$ contains an interval in $\mathcal{I}_j$. Then, for all j and each interval $c_{l..r}$ in $\mathcal{I}_j$ we maintain the points (l, r, j) in a data structure $\mathcal{D}$ that supports the following range queries: given a query point (a, b), return the highest j such that $a \leq l$ and $r \leq b$ for at least one point (l, r, j) in $\mathcal{D}$.

However, unlike the sets of intervals maintained in Section 3, our construction in this section satisfies the property that a single update affects only a *small* number of intervals in the sets $\mathcal{I}_j$ for all j. We now proceed with the technical argument.

Let S_x denote the set of positions of the element x in the sequence $c_{1..n}$. We will denote by $S_x[i]$ the position of the ith occurrence of element x. Let $I_x(l, r)$ denote the interval $c_{S_x[l]..S_x[r]}$.

Now let $\delta = 1 + \varepsilon' = (1+\varepsilon)^{1/3}$ and fix x. There are $f = F_x(c_{1..n})$ occurrences of element x in the full range $c_{1..n}$. We will maintain a subset of the $\binom{f}{2}$ possible intervals $I_x(l, r)$ in sets $\mathcal{I}_{j,x}$, $1 \leq j \leq \lceil \lg_\delta n \rceil$. We will not have need for nested intervals in $\mathcal{I}_{j,x}$; therefore, we can number each interval of $\mathcal{I}_{j,x}$ from left to right with s_k the start of interval k and e_k the end of interval k, satisfying $s_k \leq s_{k+1}$, $e_k \leq e_{k+1}$. We maintain the following two invariants on the intervals of $\mathcal{I}_{j,x}$: (1) $\delta^j \leq e_k - s_k \leq \delta^{j+1}$, and (2) $(\varepsilon'/2)\delta^j \leq s_{k+1} - s_k \leq \varepsilon'\delta^j$, and the number of positions of S_x not covered by an interval of $\mathcal{I}_{j,x}$ at either end is at most $(\varepsilon'/2)\delta^j$ (so $s_1 \leq (\varepsilon'/2)\delta^j$ and $f - r_{|\mathcal{I}_{j,x}|} \leq (\varepsilon'/2)\delta^j$). From our invariants we get the following proposition.

▶ **Proposition 6.** *An interval $I_x(l, r)$ intersects at most $2(r - l + 1)/(\varepsilon'\delta^j) + O(1/\varepsilon')$ intervals of $\mathcal{I}_{j,x}$.*

Proof. By Invariant (2), we have a gap size of between $(\varepsilon'/2)\delta^j$ and $\varepsilon'\delta^j$ elements between consecutive starting points of intervals of $\mathcal{I}_{j,x}$. Since each interval has size at most δ^{j+1}, the total number of intervals intersecting $I_x(l, r)$ is at most $2(r - l + 1 + 2\delta^{j+1})/(\varepsilon'\delta^j)$. ◀

For each interval $I_x(s_k, e_k)$ of $\mathcal{I}_{j,x}$, let $\mathtt{pot}(I_x(s_k, e_k)) = e_k - s_k + 1$ denote the number of elements of S_x (and thus positions in the original sequence $c_{1..n}$) that fall between s_k and e_k. When we insert or delete an element x, by Proposition 6, we must update the $\mathtt{pot}$ values of $O(1/\varepsilon')$ intervals of $\mathcal{I}_{j,x}$. Across all j, $1 \leq j \leq \lceil \log_\delta n \rceil$, $O((1/\varepsilon') \lg_\delta n)$ intervals are affected.

During the updates, each affected $\mathtt{pot}(I_x(s_k, e_k))$ value is incremented or decremented by one. If, for an interval $I_x(s_k, e_k)$ in $\mathcal{I}_{j,x}$, Invariant (1) is violated by the update, then we rearrange the intervals in the neighborhood of $I_x(s_k, e_k)$ as follows. Consider all intervals of $\mathcal{I}_{j,x}$ that intersect with $I_x(s_k, e_{k+1})$. By Proposition 6, there are $O(1/\varepsilon')$ such intervals. We remove said intervals and create new intervals in their place with exactly $\lceil(1 + \varepsilon'/2)\delta^j\rceil$ positions of x that fall in each interval. Furthermore, we space them so that Invariant (2) holds when the new intervals are inserted into $\mathcal{I}_{j,x}$.

To build these intervals, we must be able to efficiently search for elements by rank in S_x. As this will not dominate the update cost, we can use a typical order statistic tree, with $O(\lg f) = O(\lg n)$ query and update time. We may construct the new intervals satisfying invariants (1) and (2) with a constant number of queries on S_x per interval, thus in $O((1/\varepsilon') \lg n)$ time overall.

We can analyze the total cost of rebuilds as follows. On each update, we affect $O(1/\varepsilon')$ intervals at each level. However, the affect on $\mathtt{pot}$ is the same for each interval, and when we rebuild, we rebuild a superset of the intervals affected on update. It follows that the total amortized cost of rebuilds is $\sum_{j=1}^{\lceil \log_\delta n \rceil}(1/\delta^j) \cdot O((1/\varepsilon') \lg n) = O((\lg n)/\varepsilon'^2)$ per update.

Further, in each update we must update S_x and update the `pot` values. These take time $O(\lg n)$ and $O((1/\varepsilon')\lg_\delta n) = O(\lg n/\varepsilon'^2)$, respectively. So far we pay $O(\lg n/\varepsilon'^2)$ per update, but we have yet to describe the data structure that holds each $\mathcal{I}_{j,x}$, which will also need to be updated during rebuilds.

Consider each interval $I_x(s_k, e_k)$ of $\mathcal{I}_{j,x}$ as a point (s_k, e_k, j). We store each interval of $\mathcal{I}_{j,x}$, across all $1 \le j \le \lceil \lg_\delta n \rceil$ and all x, in a data structure $\mathcal{D}$ that supports the following range queries: given a query point (a, b), return the highest j such that $a \le l$ and $r \le b$ for at least one point (l, r, j) in $\mathcal{D}$. Associated with each point, we keep the element x from which it originated.

We first must consider the number of intervals (and thus points) stored in $\mathcal{D}$. As before, we assume element x occurs $f = F_x(c_{1..n})$ times in $c_{1..n}$. Then $|\mathcal{I}_{j,x}| = O(f/\lceil \varepsilon'\delta^j \rceil)$. Across all levels, we can bound the total number of intervals at $O(f \lg_\delta n) = O(f \lg n/\varepsilon')$ or $O(f/\varepsilon'^2)$. Accounting for all x, the number of intervals in $\mathcal{D}$ will be $O(m) = O(\min(n \lg n/\varepsilon', n/\varepsilon'^2))$.

▶ **Lemma 7.** *Data structure $\mathcal{D}$ can be stored in $O(n \lg n)$ bits, where n is the number of elements in $\mathcal{D}$. Queries and updates can be supported in $O(\lg n/\lg \lg n)$ time.*

Proof. Let P denote the set of points to be stored in our data structure. Here we use $\varepsilon > 0$ independently of the rest of the section. We start by considering the special case when the second coordinate is bounded by $\lg^\varepsilon n$, i.e., $r \le \lg^\varepsilon n$ for all $(l, r, j) \in P$. In this case it is sufficient to store $\lg n$ points for every possible value of b: let $\max_{r,j}$ denote the biggest first coordinate of a point (l', r', j') in P with $r' = r$, $j' = j$ ($\max_{r,j} = \max\{l' \mid (l', r', j') \in P$ and $r' = r, j' = j\}$. The answer to a query (a, b) is the largest j that satisfies $a \le \max_{r,j}$ for some $r \le b$. We keep all values $\max_{r,j}$ such that P contains at least one point (l, r, j) for some l, and store them in increasing order. We group them in blocks of size $\Theta(\lg^{1-\varepsilon} n)$ and we keep a global lookup table of size $o(n)$ bits that allows answering queries within any possible block.

Also, in a local lookup table of size $O(\lg^{3\varepsilon} n)$ bits we store for each block and every possible value of r the index of the block preceding it which maximizes the value of j given r. We also store a fusion tree on the values $\max_{r,j}$ so that we can compute the rank of a within these values in constant time. Given a query, we compute in constant time the block which the predecessor of a belongs to and use table lookup on that block and one other block preceding it to get the answer. Updates also take constant time since the size of individual blocks and the local lookup table fit in a single word.

A general query can be reduced to the above described special case by using a range tree with node degree $\lg^\varepsilon n$ that splits the points on the value of their second coordinate. Although every point is stored in $O(\lg n/\lg \lg n)$ nodes, our data structure uses linear space. Let $P(u)$ denote the set of points stored in a node u. We replace the second coordinate of each point $p \in P(u)$ with the index i of the child node u_i such that $p \in P(u_i)$. We keep the above described special case data structure in every node $P(u)$, but we do not store the set $P(u)$ itself. A query interval can be fully covered by $O(\lg n/\lg \lg n)$ tree nodes. We query the data structure in each one of them and return the maximum value j in $O(\lg n/\lg \lg n)$ time. Similarly, an update affects the special case data structure in $O(\lg n/\lg \lg n)$ nodes and requires $O(\lg n/\lg \lg n)$ time.

The total space usage is $O(n \log n)$ bits because we spend $O(\min(\log^{2+\varepsilon} n, |P(u)|\lg n)$ bits in each node u of the range tree. To prove this bound, we classify nodes into low and high nodes. Low nodes are the nodes in the lowest $(1 + 2/\varepsilon)$ levels of the tree and the rest of the nodes are high nodes. We also store the set of points $P(u)$ in every low node u. Thus we spend $O(|P(u)|\lg n)$ bits in every low node, so the total space consumed by all low nodes

is $O((1/\varepsilon)n \lg n)$ bits. We spend $O(\lg^{2+\varepsilon} n) = O(|P(u)|)$ bits in every high node because $|P(u)| \geq \lg^{2+\varepsilon} n$. Since the total number of points in all $P(u)$ is $O(n(\lg n/\lg \lg n))$, the total space consumed by high nodes is $O(n(\lg n/\lg \lg n))$ bits. ◀

Now suppose we are given a query range $c_{a..b}$. We find the largest j such that some interval from $\mathcal{I}_{j,x}$ for some x is contained in $c_{a..b}$. Using data structure $\mathcal{D}$ from Lemma 7, we can compute the index j in $O(\lg n/\lg \lg n)$ time. We return the element x associated with j.

▶ **Lemma 8.** *The element x returned is a $(1 + \varepsilon)$-approximate mode of query range $c_{a..b}$.*

Proof. If $c_{a..b}$ contains an interval from $\mathcal{I}_{j,x}$, then x occurs at least δ^j times in $c_{a..b}$. On the other hand, we can show that if some y occurs δ^{j+3} times in $c_{a..b}$, then $c_{a..b}$ contains an interval from $\mathcal{I}_{j+1,y}$. Recall $\delta^{j+3} = (1 + \varepsilon')\delta^{j+2}$. Each interval of $\mathcal{I}_{j+1,y}$ has size at most δ^{j+2} and there is a gap of at most $\varepsilon'\delta^j$ elements of y between the start of every interval in $\mathcal{I}_{j+1,y}$. Then since $\varepsilon'\delta^{j+1} + \delta^{j+2} < (1 + \varepsilon')\delta^{j+2}$, it must be that an interval of $\mathcal{I}_{j+1,y}$ falls in the query range $c_{a..b}$. We therefore know $\delta^j \leq F(c_{a..b}) < \delta^{j+3} = (1 + \epsilon)\delta^j$. It follows that x is a $(1 + \varepsilon)$-approximate mode of query range $c_{a..b}$. ◀

This gives us the main theorem for the section.

▶ **Theorem 9.** *There exists an $O(m \lg m)$-bit data structure, where $m = \min(n \lg n/\varepsilon, n/\varepsilon^2)$ that answers $(1 + \varepsilon)$-approximate range mode queries in $O(\lg m/\lg \lg m)$ time. Insertions and deletions are supported in $O(\lg n/\varepsilon^2)$ time.*

Proof. We have $(1 + \varepsilon')^3 = (1 + \varepsilon)$ and $(1 + \varepsilon)^3 = \varepsilon^3 + 3\varepsilon^2 + 3\varepsilon + 1$. The smallest exponent dominates $O(1/\varepsilon')$ since $\varepsilon \leq 1$ and thus $\varepsilon' < \varepsilon \leq 1$. Thus we have $1/\varepsilon' = O(1/\varepsilon)$. As previously stated, the number of intervals in $\mathcal{D}$ is $O(m)$, where $m = \min(n \lg n/\varepsilon, n/\varepsilon^2)$. The space bound for $\mathcal{D}$ is thus $O(m \lg m) = \Omega(n \lg n)$ bits, which dominates the total space cost. The query time is $O(\lg m/\lg \lg m)$.

The update cost has four components: Updating $\mathcal{D}$, updating S_α, and updating `pot` values for all affected intervals. As previously mentioned, the latter three are dominated by $O(\lg n/\varepsilon'^2) = O(\lg n/\varepsilon^2)$. Via Lemma 7, the cost of updating $\mathcal{D}$ is $O(\lg m/\lg \lg m)$. Since m is no more than n/ε^2, $\lg m/\lg \lg m$ is dominated by $O(\lg n/\varepsilon^2)$. In total, the cost of updates is $O(\lg n/\varepsilon^2)$. ◀

We can use our dynamic data structure to obtain a result for approximate range mode queries on two-dimensional points. Our data structure can find approximate mode in the case when the query range is bounded on three sides.

▶ **Corollary 10.** *There exists a data structure that supports three-sided two-dimensional approximate range mode queries in $O(\log m)$ time and uses $O(m \log m)$ words of $\log n$ bits, where $m = \min(n \lg n/\varepsilon, n/\varepsilon^2)$.*

Proof. Using the technique introduced by Dietz in [6], we can transform a data structure that supports updates in $u(n)$ time and queries in $q(n)$ time into an offline partially persistent data structure that answers queries in $O(q(n) \cdot \log \log n)$ time and uses $O(n \cdot u(n))$ words of space. Using sweep line technique, we can transform an offline partially persistent data structure for one-dimensional queries into a static data structure for three-sided queries with the same time and space bounds. ◀

5   Approximate Range Median and Range Selection

In this section we present solutions to approximate range selection queries. As discussed previously, a range selection query takes two indices a, b of a sequence $c_1, \ldots, c_n$ and must return the index of an element x whose rank in $c_{a..b}$ is between $k - \alpha(b - a + 1)$ and $k + \alpha(b - a + 1)$. We study two variants. In the first variant, the rank k is supplied prior to construction of the data structure. In the second variant, we allow k to be specified at query time. Here rank is defined so the ith smallest element in the range has rank i. We also support a specific k depending on the size of the range, i.e. $f(b - a + 1) = \lceil (b - a + 1)/2 \rceil$, which is the range median problem. We make the restrictions $f(x) \le f(x + 1) \le f(x) + 1$ and $1 \le f(x) \le x$.

▶ **Theorem 11.** *Any one-dimensional approximate range median data structure requires* $\Omega(n)$ *bits.*

Proof. Assume n is even. Divide the sequence S of size n to $n/2$ blocks each of size 2. We say that S satisfies property $(*)$ if for each block b in S one of the following two conditions hold:
- either b consists of $\{1, 2\}$,
- or b consists of $\{2, 1\}$.

Clearly, the number of sequences that satisfy $(*)$ is $2^{(n/2)}$ since there exists $n/2$ blocks in a sequence of size n and each block can have one of two different values. Moreover for any two distinct sequences S_1 and S_2 satisfying $(*)$ differing at block b, the approximate range selection query must be exact on block b, and therefore must return different values. Thus, the information theoretic lower bound for storing an approximate range median data structure is $\Omega(\lg 2^{(n/2)}) = \Omega(n/2) = \Omega(n)$ bits.   ◀

Fixed Rank $f(b - a + 1) = k$ Selection.   We first address the range selection variant with a fixed rank $f(b - a + 1) = k$. We use a similar approach to the one in Lemma 4. We again assume n is a power of 2. At the top level, we store values $m_{n/2,i,j}$ $(1 \le i, j \le \lceil \lg_{1+\alpha} n \rceil)$. Let $r_i = n/2 - (1 + \alpha)^i$ and $s_j = n/2 + 1 + (1 + \alpha)^j$. Then $m_{n/2,i,j}$ is the element of rank $f(s_j - r_i + 1)$ in the range $c_{\lfloor r_i \rfloor .. \lceil s_j \rceil}$. We then build the structure recursively on the left and right halves of the full range.

Given a query range $c_{a..b}$, we find the appropriate element $m_{t,i,j}$ where $a \le t$, $t + 1 \le b$, and i and j are largest possible satisfying $a \le r_i$ and $s_j \le b$. We return $m_{t,i,j}$.

▶ **Lemma 12.** *The above data structure returns an α-approximate fixed-rank k element of any query range $c_{a..b}$.*

Proof. Let $x = r_i - a$ and $y = b - s_j$. Consider the size of x. If we let $z = (1 + \alpha)^i$, then $x + z < (1 + \alpha)z$. It follows $x < \alpha z$. Since $z \le (t - a + 1)$, and applying similarly for y, we can show $x < \alpha(t - a + 1)$ and $y < \alpha(b - t)$. The elements in the ranges represented by x and y shift the true rank k element of $c_{a..b}$ at most $x + y < \alpha(b - a + 1)$ ranks from $m_{t,i,j}$. It follows that $m_{t,i,j}$ is an α-approximate rank k element for range $s_a, \ldots, s_b$.   ◀

As for Theorem 4, to find the level to query, we find the highest set bit of a XOR b, then find the appropriate index $m_{t,i,j}$ via arithmetic. In total, the query takes constant time.

We now analyze the space required. At the top level, we use $O(\lg^2_{1+\alpha}(n) \cdot \lg n)$ bits, which is equal to $O(\frac{\lg^3 n}{\lg^2(1+\alpha)}) = O(\lg^3 n/\alpha^2)$ bits. Therefore our recurrence is $S(n) = 2S(n/2) + O(\lg^3 n/\alpha^2)$. The recursion tree is leaf-heavy, with total space amounting to $O(n/\alpha^2)$ bits.

▶ **Theorem 13.** *There exists an $O(n/\alpha^2)$-bit data structure that supports one-dimensional α-approximate fixed-rank $f(b - a + 1) = k$ selection queries in constant time.*

Online Rank k Selection. Our data structure from the previous section can be adapted to support queries that specify the rank k at query time. We again assume n is a power of 2. Let $\delta = 1 + \alpha/2$. At the top level we now store values $m_{n/2,i,j,l}$ $(1 \le i, j, \le \lceil \lg_\delta n \rceil, 0 \le l \le \lfloor 1/\alpha \rfloor)$. Again, we let $r_i = n/2 - \delta^i$ and $s_j = n/2 + 1 + \delta^j$. However, this time, $m_{n/2,i,j,l}$ represents the element of rank $q_l = l\alpha \cdot (s_j - r_i + 1) + 1$ in $c_{r_i..s_j}$. As q_l may be fractional, for simplicity we just store both rank $\lfloor q_l \rfloor$ and $\lceil q_l \rceil$ elements. We build this structure recursively on both halves of the full range.

Given a query $s_{a..b}$, we again find the appropriate element $m_{t,i,j,l}$ where $a \le t$, $t + 1 \le b$, i and j are largest possible satisfying $a \le r_i$ and $s_j \le b$, and l is chosen so q_l is as close to k as possible. We return $m_{t,i,j,l}$.

▶ **Lemma 14.** *The above data structure returns an α-approximate rank k element of any query range $c_{a..b}$ and specified rank k.*

Proof. Again let $x = r_i - a$ and $y = b - s_j$. For the same reasons as in the proof of Lemma 12, we have $x + y < \alpha(b - a + 1)/2$.

There are no more than $\alpha \cdot (s_j - r_i + 1) \le \alpha \cdot (b - a + 1)$ ranks between each consecutive q_l and q_{l+1}. Thus our chosen q_l satisfies $|q_l - k| < \lfloor \alpha(b - a + 1)/2 \rfloor$. It follows that $m_{t,i,j,l}$ is no more than $\alpha \cdot (b - a + 1)$ ranks away from the true rank k element in range $c_{a..b}$.   ◀

The query time follows as in the previous section. However, we must account for the additional space usage. Our recurrence is now $T(n) = 2T(n/2) + O(\lg^3 n/\alpha^3)$, from the additional $1/\alpha$ factor in the space cost at each level. This totals to $O(n/\alpha^3)$ bits.

▶ **Theorem 15.** *There exists an $O(n/\alpha^3)$-bit data structure that supports one dimensional α-approximate online rank k selection queries in constant time.*

───── **References** ─────

1   Prosenjit Bose, Evangelos Kranakis, Pat Morin, and Yihui Tang. Approximate Range Mode and Range Median Queries. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science*, pages 377–388, 2005.

2   Gerth Stølting Brodal, Beat Gfeller, Allan Grønlund Jørgensen, and Peter Sanders. Towards optimal range medians. *Theoretical Computeer Science*, 412(24):2588–2601, 2011.

3   Gerth Stølting Brodal and Allan Grønlund Jørgensen. Data Structures for Range Median Queries. In *Proceedings of the 20th International Symposium on Algorithms and Computation*, pages 822–831, 2009.

4   Timothy M Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T Wilkinson. Linear-space data structures for range mode query in arrays. *Theory of Computing Systems*, 55(4):719–741, 2014.

5   Timothy M. Chan and Bryan T. Wilkinson. Adaptive and Approximate Orthogonal Range Counting. *ACM Trans. Algorithms*, 12(4):45:1–45:15, 2016.

6   Paul F. Dietz. Fully Persistent Arrays (Extended Array). In *Proceedings of Workshop on Algorithms and Data Structures (WADS)*, pages 67–74, 1989. `doi:10.1007/3-540-51542-9_8`.

7   Hicham El-Zein, Meng He, J. Ian Munro, and Bryce Sandlund. Improved Time and Space Bounds for Dynamic Range Mode. In *Proceedings of the 26th Annual European Symposium on Algorithms*, pages 25:1–25:13, 2018.

8   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.

9   M. L. Fredman and D. E. Willard. BLASTING Through the Information Theoretic Barrier with FUSION TREES. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 1–7, 1990.

10 Travis Gagie, Simon J. Puglisi, and Andrew Turpin. Range Quantile Queries: Another Virtue of Wavelet Trees. In *Proceedings of the 16th International Symposium on String Processing and Information Retrieval*, pages 1–6, 2009.

11 Pawel Gawrychowski and Patrick K. Nicholson. Optimal Encodings for Range Top- k k , Selection, and Min-Max. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming*, pages 593–604, 2015.

12 Beat Gfeller and Peter Sanders. Towards Optimal Range Medians. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming*, pages 475–486, 2009.

13 Mark Greve, Allan Grønlund Jørgensen, Kasper Dalgaard Larsen, and Jakob Truelsen. Cell Probe Lower Bounds and Approximations for Range Mode. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming*, pages 605–616, 2010.

14 Roberto Grossi, John Iacono, Gonzalo Navarro, Rajeev Raman, and S. Srinivasa Rao. Asymptotically Optimal Encodings of Range Data Structures for Selection and Top-k Queries. *ACM Transactions on Algorithms*, 13(2):28:1–28:31, 2017.

15 Meng He, J. Ian Munro, and Patrick K. Nicholson. Dynamic Range Selection in Linear Space. In *Proceedings of the 22nd International Symposium*, pages 160–169, 2011.

16 Allan Grønlund Jørgensen and Kasper Green Larsen. Range Selection and Median: Tight Cell Probe Lower Bounds and Adaptive Data Structures. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 805–813, 2011.

17 Danny Krizanc, Pat Morin, and Michiel H. M. Smid. Range Mode and Range Median Queries on Lists and Trees. *Nord. J. Comput.*, 12(1):1–17, 2005.

18 M. Patrascu. Succincter. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 305–313, 2008.

19 Holger Petersen. Improved Bounds for Range Mode and Range Median Queries. In *Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science*, pages 418–423, 2008.

20 Holger Petersen and Szymon Grabowski. Range mode and range median queries in constant time and sub-quadratic space. *Information Processing Letters*, 109(4):225–228, 2009.