Linear-Space Data Structure for Parametrized Range Mode Query in Arrays*

Ovidiu Rața[1], Paul Flavian Diac[1]

Alexandru Ioan Cuza Universy of Iași,
Faculty of Computer Science, Iași, Romania

Abstract. We present a data structure for the range mode query problem, studied by Chan et al [1], He,Liu.[3] and Xu, Williams et al[2], which requires $O(n)$ space, and answers queries on an interval $[i, j]$ in parametrized $O(\sqrt{j-i+1})$ time. The standard RAM model is assumed, with word size $w = \Theta(\log n)$.

Additionally, we present a linear-space data structure, that requires $O(\min(\sqrt{j-i+1}, \sqrt{j/w}))$ parametrized time per query, and supports element insertion at the end of the array $A[1:n]$ in amortyzed $O(w + \sqrt{n \cdot w})$ time, by improving over the method proposed by Chan et al[1], using compact rank/select data structures that support appending an element at the end of the binary array in amortyzed $O(1)$ time.

Keywords: Data structure · Parametrized algorithms · Range queries · Mode.

1 Introduction

2 Finding a Range Mode

Our data structure is constructed by extending the ideas of Chan et al[1].

Data Structure precomputation. Let D denote the set of elements of the array A, and assume some arbitrary ordering on the elements. We will maintain D, as a red-black tree. We denote the number of distinct elements of A as Δ. First, we apply rank-space reduction, and construct the array B, such that, for each i, $B[i]$ stores the rank of $A[i]$ in D. Thus, $B[i] \in \{1, \ldots, \Delta\}$. Let C be an array of size Δ, that provides a direct mapping from $\{1, \ldots, \Delta\}$ to D. Set D and arrays B and C can be computed inf $O(n \log \Delta)$ time. Further, we will focus on finding a range mode x, over array B, which will be transformed into the respective range mode over array A, using array C, that is, $y = C[x]$. For each $a \in \{1, \ldots \Delta\}$, let $Q_a = \{b \mid B[b] = a\}$. We will represent the sets Q_a as ordered arrays. Also, we define the array Q^{-1}, of size n, s.t. $\forall i \in \{1, \ldots, n\}, Q_i^{-1} = k, Q_{B_i}(k) = i$.

* Supported by Alexandru Ioan Cuza Universy of Iași.

Additional Definitions. Let $\phi(i,j)$ be the function that returns the frequency of the most frequent element in the range $B[i:j]$.

Let $\mathbf{freq}_x(i,j)$, be the frequency of element x of array B, inside the interval $[i,j]$.

Further, we will show how to build a data structure, that supports parametrized range mode query in time $O(\sqrt{j-i+1})$ for a query interval $[i,j]$.

Lemma 1. *Given an array $A[1:n]$, there exists a data structure, requiring arrays B, C, Q, Q^{-1}, the set D, and $O(n/w)$ additional words of RAM, that for some fixed integer $L \in \{0,\cdots,\lceil \log n \rceil\}$, for intervals $[i,j]$ s.t. $2^{L-1} < j-i+1 \leq 2^L$ supports queries of the following form, in $O(\sqrt{2^L})$ time:*

For the interval $[i,j]$, determine an element x, s.t. $\boldsymbol{freq}_x(i,j) \geq \min(\phi(i,j),\sqrt{2^L})$

Proof. First, we fix an integer $L \in \{0,\cdots,\lceil \log n \rceil\}$. Then, we separate the array $A[1:n]$ into adjacent blocks, of length $b_L = \sqrt{2^L}$. The block $i \in \{1,\ldots,\lfloor \frac{n}{b_L} \rfloor\}$ encompasses the interval $[(i-1)\cdot b_L + 1, \min(i \cdot b_L, n)]$.

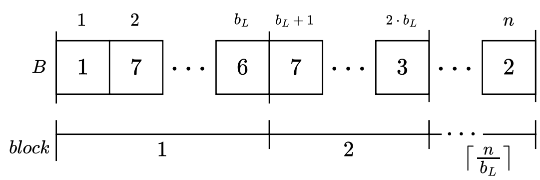

Fig. 1. Example of separating an array B into $\lceil \frac{n}{b_L} \rceil$ adjacent blocks of length $b_L = \sqrt{2^L}$.

Let s_i^L, be a binary string, similar to the string defined in the data structure proposed by Chan et al.[1]:

Definition 1. *For integer $L \in \{0,\cdots,\lceil \log n \rceil\}$, and block i of size $\sqrt{2^L}$, let s_i^L be a binary string, obeying the following properties:*

Property 1. *Consider the interval of blocks, from block i to block $j = \min(\lfloor \frac{n}{b_L} \rfloor, i + \sqrt{2^L} - 1)$. There will be $j-i+1 = O(\sqrt{2^L})$ bits with value 1 in s_i^L, the k-th set bit corresponding to the right border of the interval k.*

Property 2. *For any integer $k \in [i,j]$, consider the interval of blocks $[i,k]$.*

For the interval of blocks $[i,k]$, consider the endpoints of array A which correspond to the left endpoint of the block i, and the right endpoint of the block k to be $i' = (i-1)\cdot b_L + 1, k' = k \cdot b_L$.

In the string s_i^L, there are exactly $\min(\sqrt{2^L}, \phi(i',k'))$ bits with value 0 to the left of the k-th set bit.

Note that the length of the binary string s_i^L is at most $2 \cdot \sqrt{2^L} = O(\sqrt{2^L})$.

Every string s_i^L, can be represented with a succint or compact data structure that supports rank-select operations, in $O(\sqrt{2^L}/w)$ RAM words.

There are $\lfloor \frac{n}{2^L} \rfloor$ blocks of length exactly b_L, thus, the space necessary for maintaining the strings s_i^L is:

$$\lfloor \frac{n}{2^L} \rfloor \cdot O(\sqrt{2^L}/w) = O(n/w) \text{ words of RAM.}$$

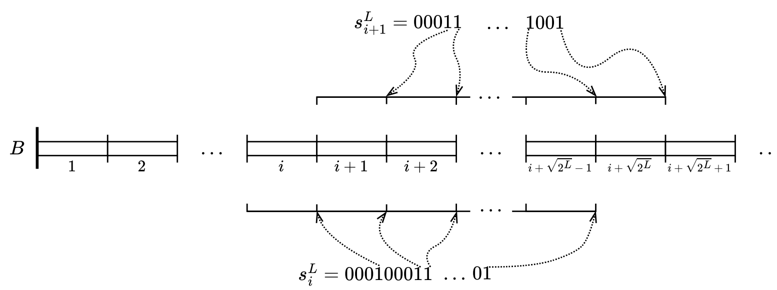

Fig. 2. Example of strings s_i^L and s_{i+1}^L over an array B, separated into blocks of size b_L. Each bit with value 1 in the strings s_i^L and s_{i+1}^L represents the right border of some block.

Query Algorithm. The query alorithm is similar to that of the data structure proposed by Chan et al.[1]:

Given the interval $[i, j]$:

1. If $j - i + 1 \leq 2 \cdot \sqrt{2^L}$, then just iterate through the elements of $[i, j]$ in $O(\sqrt{2^L})$ time, or determine the blocks that are entirely contained inside of $[i, j]$. Let the block and the last blocks be β_1 and β_2 respectively.
2. There will be at most $\sqrt{2^L}$ blocks from block β_1 to block β_2.
   Let $[i', j']$ be the interval of A, which corresponds to blocks β_1 through β_2. We will use the string $s_{\beta_1}^L$ to get the frequency f, of an element from $[i', j']$ s.t. $f \geq \min(\phi(i', j'), \sqrt{2^L})$. This can be done by select operations over $s_{\beta_1}^L$, more precisely, we will determine $p = \mathbf{select}_1(s_{\beta_1}^L, \beta_2 - \beta_1 + 1)$, which is the position in the string $s_{\beta_1}^L$, corresponding to the right endpoint of the block β_2, with respect to block β_1. The value f, corresponding to the number of bits with value 0 in the interval $[1, p_2]$ of the string $s_{\beta_1}^L$, can be calculated as follows:
   $$f = (p) - (\beta_2 - \beta_1 + 1)$$
   We can further use binary search, to get the number β_f and the position p_f in string $s_{\beta_1}^L$ of the first block, such that there are exactly f bits with value 0

in the interval $[1, p_f]$ of the string $s^L_{\beta_1}$. Further, we can iterate through each position k, of the block β_f, and use arrays Q_{B_k} in order to determine an element x with frequency at least f. (Chan et al.[1])
This procedure will take $O(\sqrt{2^L})$ time, as the most time consuming step is the iteration through elements of β_f.
We must also note, that this step will yield exactly the answer for the query (i', j').

3. Further, we can iterate through each position $k \in ([i,j] \setminus [i',j'])$, and increase f every time we can if it is $< \sqrt{2^L}$, by checking whether $Q_{B_k}(Q_k^{-1} \pm (f+1)) \in [i,j]$. This will clearly take $O(\sqrt{2^L})$ time, as there will be $O(\sqrt{2^L})$ elements in the prefix and suffix of $[i,j]$, and we will increase f at most $O(\sqrt{2^L})$.
This step clearly determines the answer for the query $[i,j]$, as it uses the value f characterising the answer for the query (i',j'), and uses the prefix and suffix of $[i,j]$, in order to adapt f to the answer for the query (i,j). These steps are the same as in the data structure proposed by Chan et al.[1], and proofs for their correctness are also provided in their work.

Finally, the required element x of the range $B[i:j]$, can be transformed to the corresponding element of A, which is C_x. Thus, the answer to the query (i,j) given above, is calculated in $O(\sqrt{2^L})$ time.□

Algorithm 1 Lemma 1 Query Algorithm

function QUERY
▷ Step 1:
  if $j - i + 1 \leq 2 \cdot \sqrt{2^L}$ **then**
    #Iterate through each element of [i,j] and return the answer.
  $\beta_1 \leftarrow \lfloor \frac{i-1}{\sqrt{2^L}} \rfloor + 1, \beta_2 \leftarrow \lfloor \frac{j}{\sqrt{2^L}} \rfloor$
  $i' \leftarrow (\beta_1 - 1) \cdot \sqrt{2^L} + 1, j' \leftarrow \beta_2 \cdot \sqrt{2^L}$
▷ Step 2:
  $p \leftarrow \mathbf{select}_1(s^L_{\beta_1}, \beta_2 - \beta_1 + 1), f \leftarrow (p) - (\beta_2 - \beta_1 + 1)$
  # Find β_f and p_f, and iterate through the block β_f, and get the candidate element x.
▷ Step 3:
  for $k \in [i, i'-1]$ **do**
    while $Q_{B_k}(Q^{-1}(k) + f + 1) \leq j$ and $f < \sqrt{2^L}$ **do**
      $f \leftarrow f + 1$
      $x \leftarrow B_k$
  for $k \in [j'+1, j]$ **do**
    while $Q_{B_k}(Q^{-1}(k) - f - 1) \geq i$ and $f < \sqrt{2^L}$ **do**
      $f \leftarrow f + 1$
      $x \leftarrow B_k$
  return $(C(x), f)$

Further, for any integer $L \in \{0, \cdots, \lceil \log n \rceil\}$ we will define $b'_L = 2^L$, as the size of the big blocks, and will separate the array $B[1:n]$ into adjacent blocks of size b'_L. If we cannot exactly divide the array $B[1:n]$ into blocks of size b'_L, then the last elements of array $B[1:n]$ will just be included into a last block, of size $< b'_L$.

For every block $i \in \{1, \ldots, \lceil \frac{n}{b'_L} \rceil\}$ of size b'_L, we denote by F_i^L, the set of elements in the range $B[(i-1) \cdot b'_L + 1 : \min((i+1) \cdot b'_L, n)]$, that have frequency $> \sqrt{2^L}$. Note that the interval $[(i-1) \cdot b'_L + 1 : \min((i+1) \cdot b'_L, n)]$ contains 2 big blocks, if its size is not limited by n.

If we maintain the sets F_i^L as ordered arrays, then the amount of space required for storing F_i^L will be:

$$\sum_{L=0}^{\lceil \log n \rceil} \sum_{i=1}^{\lceil \frac{n}{b'_L} \rceil} |F_i^L| \leq \sum_{L=0}^{\lceil \log n \rceil} \sum_{i=1}^{\lceil 2 \cdot \frac{n}{b'_L} \rceil} 2 \cdot \sqrt{2^L}$$

$$= O(\sum_{L=0}^{\inf} \sum_{i=1}^{\lfloor \frac{n}{2^L} \rfloor} \sqrt{2^L}) = O(\sum_{L=0}^{\inf} \lfloor \frac{n}{2^L} \rfloor \cdot \sqrt{2^L})$$

$$= O(\sum_{L=0}^{\inf} \frac{n}{\sqrt{2^L}}) = O(\sum_{L=0}^{\inf} \frac{n}{\sqrt{2^L}}) = O(n \cdot \sum_{L=0}^{\inf} \frac{1}{\sqrt{2^L}})$$

$$= O(n \cdot \frac{\sqrt{2}}{\sqrt{2}-1}) = O(n)$$

Thus, storing the sets F_i^L is $O(n)$ words of RAM, as storing each element of F_i^L requires 1 word of RAM.

Lemma 2. *Given an array $A[1:n]$, there exists a data structure, requiring arrays B, C, Q, Q^{-1}, F_i^L, the set D, and $O(n/w)$ additional words of RAM, that for some fixed integer $L \in \{0, \cdots, \lceil \log n \rceil\}$, for intervals $[i,j]$ s.t. $2^{L-1} \leq j-i+1 < 2^L$ and supports queries of the following form, in $O(\sqrt{2^L})$ time:*

For the range $A[i:j]$, determine an element of frequency $\phi(i,j)$, if $\phi(i,j) > \sqrt{2^L}$, or -1 if no such element exists.

Proof. First, we fix an integer $L \in \{0, \cdots, \lceil \log n \rceil\}$. Then, we separate the array $A[1:n]$ into adjacent blocks, of length $b'_L = 2^L$. The block $i \in \{1, \ldots, \lceil \frac{n}{b'_L} \rceil\}$ encompasses the interval $[(i-1) \cdot b'_L + 1, \min(i \cdot b'_L, n)]$.

For each element x in F_i^L, we define the binary string $Y_{i,x}^L$, the following way:

Definition 2. *For integer $l \in \{0, \cdots, \lceil \log n \rceil\}$, for each block i of size 2^L, for each element $x \in F_i^L$, let $Y_{i,x}^L$ be a binary string, obeying the following properties:*

*Property 3. The string $Y_{i,x}^L$ will have at most $2 \cdot \sqrt{2^L}$ bits set to value 1. Each of these bits, will correspond to the right endpoint of a small block of size $b_L = \sqrt{2^L}$, declared in **lemma 1**, that lies inside the big block i, of size b'_L, or in the big block $i+1$.*

Property 4. For every k, from 1 to $2 \cdot \sqrt{2^L}$, let $k' = \min(k' \cdot b_L + i', n)$ be the right endpoint of the k-th small block, counting from the first small block inside of the big block i, or just n, if such a block does not exist. Note that the small block k may lie inside the big block $i+1$.

Inside the string $Y^L_{i,x}$, there will be exactly $\boldsymbol{freq}_x(i', k')$ bits with value 0 to the left of the k-th bit with value 1.

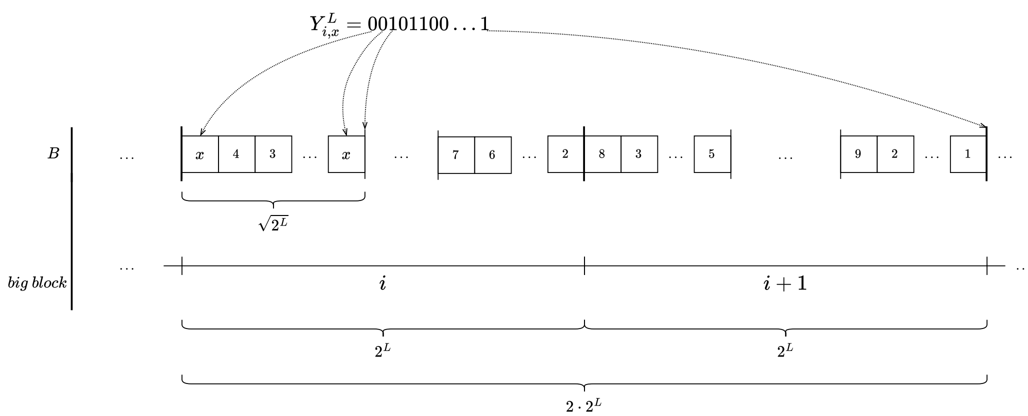

Fig. 3. Example of string $Y^L_{i,x}$ over an array B, for an element $x \in F^L_i$. The array B separated into big blocks of size b'_L. The big block i is in its turn separated into small blocks of size b_L. Each bit with value 1 in string $Y^L_{i,x}$ corresponds to the right border of a small block of size b_L, and each bit with value 0 corresponds to an encounter of the element x in the array B.

The string $Y^L_{i,x}$ will have exactly $\mathbf{size}_0 = \mathbf{freq}_x(b'_L \cdot (i-1)+1, \min(b'_L \cdot (i+1), n))$ bits with value 0. Note that $\mathbf{size}_0 > \sqrt{2^L}$. Also, there will be exactly $\mathbf{size}_1 = 2 \cdot \sqrt{2^L}$ bits with value 1. Thus, the size of the string will be : $\mathbf{size}_0 + \mathbf{size}_1 \leq 3 \cdot \mathbf{size}_0$.

Thus, the total length of the strings $Y^L_{i,x}$, over the blocks $i \in \{1, \ldots, \lceil \frac{n}{b'_L} \rceil\}$ will be:

$$\sum_{i=1}^{\lceil \frac{n}{b'_L} \rceil} \sum_{x \in F^L_i} |Y^L_{i,x}| \leq \sum_{i=1}^{\lceil \frac{n}{b'_L} \rceil} \sum_{x \in F^L_i} 3 \cdot \mathbf{size}_0$$

$$= \sum_{i=1}^{\lceil \frac{n}{b'_L} \rceil} \sum_{x \in F^L_i} 3 \cdot \mathbf{freq}_x(b'_L \cdot (i-1)+1, \min(b'_L \cdot (i+1), n))$$

$$\leq \sum_{x \in F^L_i} 3 \cdot 2 \cdot \mathbf{freq}_x(1, n) \leq \sum_{x \in B[1:n]} 6 \cdot \mathbf{freq}_x(1, n) = O(n)$$

The total length of the strings $Y^L_{i,x}$ will be $O(n)$ bits, thus, for a fixed L, they will require $O(n/w)$ words of RAM to be stored, and to support rank/select operations in $O(1)$.

Query Algorithm. The query alorithm is as follows:

Given the interval $[i, j]$:

1. If $j - i + 1 \leq 2 \cdot \sqrt{2^L}$, then just iterate through the positions of $B[i, j]$ and determine the answer. Otherwise, the interval $[i, j]$ will intersect, at most 2 big blocks of size b'_L. Let these blocks be β'_1 and β'_2. Let the interval of small blocks that are entirely covered by the interval $[i, j]$, be $[\beta_1, \beta_2]$. These blocks can be determined rapidly, as $\beta_2 = \lfloor j/b_L \rfloor$ and $\beta_1 = \lfloor (i + b_L)/b_L \rfloor$, and similarly for the big blocks β'_1 and β'_2.
2. Further, we iterate through each element $x \in F^L_{\beta'_1}$, and determine :

$$p^x_1 = \mathbf{select}_1(Y^L_{\beta'_1,x}, \beta_1 - (\beta'_1 - 1) \cdot \frac{b'_L}{b_L} - 1), p^x_2 = \mathbf{select}_1(Y^L_{\beta'_1,x}, \beta_2 - (\beta'_1 - 1) \cdot \frac{b'_L}{b_L})$$

$$f_x = (p^x_2 - p^x_1) - (\beta_2 - \beta_1 + 1), f = \max_{x \in F^L_{\beta'_1}} (f_x)$$

3. Now, we iterate through the positions k of the prefix and the suffix of the interval $[i, j]$, which are not contained in any of the small blocks in the interval$[\beta_1, \beta_2]$, and check wether we can increase f by 1 or not, by verifying if $Q_{B_k}(Q^{-1}_k \pm (f + 1)) \in [i, j]$.
4. Finally, if f will achieve a value $> \sqrt{2^L}$, then return f, or -1 otherwise.

Analysis. We can clearly see, that the runtime of the query algorithm will be $O(\sqrt{2^L})$, as the first step requires $O(1)$ time, the second step requires $O(\sqrt{2^L})$ time, as at most $O(\sqrt{2^L})$ elements of the set F^L_i will be verified and the **select** operations will take $O(1)$ time. The third step will also take $O(\sqrt{2^L})$ time, as there will be $O(\sqrt{2^L})$ elements of the prefix and the suffix which will have to be verified, and the value of f will be increases by 1 at most $O(\sqrt{2^L})$ times. Note that the element x with maximum frequency, can also be easily tracked during each step.

Algorithm 2 Lemma 2 Query Algorithm

function QUERY
▷ Step 1:
  if $j - i + 1 \leq 2 \cdot \sqrt{2^L}$ **then**
    #Iterate through each element of [i,j] and return the answer.
  $b_L \leftarrow \sqrt{2^L}, b'_L \leftarrow 2^L$
  $\beta_1 \leftarrow \lfloor \frac{i-1}{b_L} \rfloor + 1, \beta_2 \leftarrow \lfloor \frac{j}{b_L} \rfloor$
  $\beta'_1 \leftarrow \lfloor \frac{i-1}{b'_L} \rfloor + 1, \beta'_2 \leftarrow \lfloor \frac{j}{b'_L} \rfloor$
  $i' \leftarrow (\beta_1 - 1) \cdot \sqrt{2^L} + 1, j' \leftarrow \beta_2 \cdot \sqrt{2^L}$
  $f \leftarrow -1, m \leftarrow -1$
▷ Step 2:
  for $x \in F^L_{\beta'_1}$ **do**
    $p^x_1 \leftarrow \mathbf{select}_1(Y^L_{\beta'_1,x}, \beta_1 - (\beta'_1 - 1) \cdot \frac{b'_L}{b_L} - 1)$
    $p^x_2 \leftarrow \mathbf{select}_1(Y^L_{\beta'_1,x}, \beta_2 - (\beta'_1 - 1) \cdot \frac{b'_L}{b_L})$
    $f_x \leftarrow (p^x_2 - p^x_1) - (\beta_2 - \beta_1 + 1)$
    if $f_x > f$ **then**
      $f \leftarrow f_x$
      $m \leftarrow x$
  if $m = -1$ **then return** $(-1, -1)$
▷ Step 3:
  for $k \in [i, i' - 1]$ **do**
    while $Q_{B_k}(Q^{-1}(k) + f + 1) \leq j$ **do**
      $f \leftarrow f + 1$
      $m \leftarrow B_k$
  for $k \in [j' + 1, j]$ **do**
    while $Q_{B_k}(Q^{-1}(k) - f - 1) \geq i$ **do**
      $f \leftarrow f + 1$
      $m \leftarrow B_k$
▷ Step 4:
  if $f \geq \sqrt{2^L}$ **then return** $(C(m), f)$
  else return $(-1, -1)$

Further, we will use the results from **lemma 1** and **lemma 2** to prove that a $O(\sqrt{j-i})$ runtime per query is possible, by using $O(n)$ space.

Theorem 1. *Given an array $A[1:n]$, there exists a data structure, requiring arrays B, C, Q, Q^{-1}, the set D, and additional $O(n)$ words of RAM, that supports range mode queries over an interval $[i,j]$ in time $O(\sqrt{j-i+1})$.*

Proof. Firstly, we will build the arrays B, C, Q, and Q^{-1}, which will take $O(n \log n)$ time and $O(n)$ space. Also, for each $L \in \{0, \dots, \lceil \log n \rceil\}$, we will build F^L_i. It is easy to see, that this will take $O(n \log n)$ time, and $O(n)$ space.

Further, for each $L \in \{0, \dots, \lceil \log n \rceil\}$, we will build an instance of the data structure described in **lemma 1**, denoted by DS^L_1, and an instance of the data structure described in **lemma 2**, denoted by DS^L_2. These instances will take

$O(n)$ space, as the arrays B, C, Q, Q^{-1} and F_i^L will be shared among them, and the additional information will take $O(\log n \cdot n/w) = O(n)$ space.

Query Algorithm. The query algorithm is as follows:

Given the interval $[i, j]$:
1. Determine the smallest L, s.t $2^{L-1} < (j - i + 1) \leq 2^L$. Note that $2^L = O(j - i + 1)$.
2. We will determine $(f, x) = \max(DS_1^L.query(i, j), DS_2^L.query(i, j))$, where $DS_1^L.query(i, j)$, and $DS_2^L.query(i, j)$ denotes querying the instances DS_1^L and DS_2^L, respectively, for the interval $[i, j]$. (f, x) will be exactly the answer for the query (i, j), as $f = \phi(i, j)$, and x is an element of the array A, s.t. $\exists y, s.t.\ (C_y = x) \wedge (\mathbf{freq}_y(i, j) = f)$.

Analysis. The algorithm will take $O(\sqrt{2^L}) = O(\sqrt{j - i + 1})$ time, as $2^L = O(j - i + 1)\square$.

Lemma 3. *Given an array $A[1 : n]$, there exists a data structure requiring $O(n)$ words of RAM, that supports range mode queries over an interval $[i, j]$ in $O(\sqrt{j/w})$ time.*

Proof. Firstly, we build the arrays B, C, Q, Q^{-1}, and set D, as in the previous data structures.

In their work, Chan et al.[1] have divided the array $A[1 : n]$ into blocks of equal length, in order to prove their result. We will take a similar approach, one difference being that we will divide the array $A[1 : n]$ into $O(\sqrt{w \cdot n})$ blocks of varying size.

Firstly, we divide the array $A[1 : n]$ into adjacent big blocks, block i (counting from left to right) having size i. It is easy to see, that there will be at most $O(\sqrt{n})$ big blocks. Further, we will divide each big block i, into small blocks, of size $i/\sqrt{w}$.

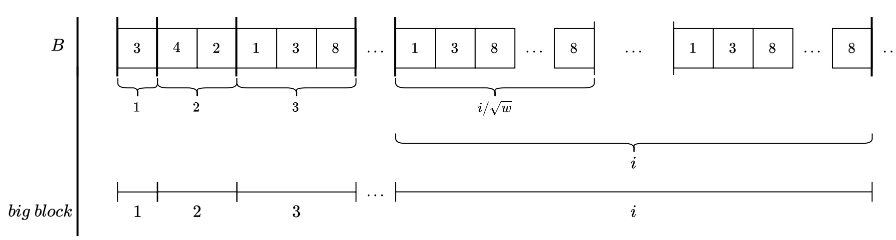

Fig. 4. Example of separating the array B into big blocks. The big bocks $1, 2, 3$ have sizes $1, 2, 3$ respectively. The i-th big block has size i. The i-th big block is in its turn divided into small blocks of size $i/\sqrt{w}$.

Let the number of small blocks be b. For each small block β we will store the string z_β and the set H_β, defined below:

Definition 3. *For each small block $\beta \in \{1, \ldots, b\}$, let z_β be a binary string obeying the following properties:*

Property 5. There are exactly β bits with value 1 in string z_β. The k-th bit with value 1 counting from left to right will correspond to the left border of the k-th small block.

Property 6. For each bit k with value 1, let l_k be the position of the left endpoint of the small block k in array B, and let r_β be the position of the right endpoint of the small block β in array B. There will be exactly $\min(\phi(l_k, r_\beta), b)$ bits with value 0, to the right of the k-th bit with value 1.

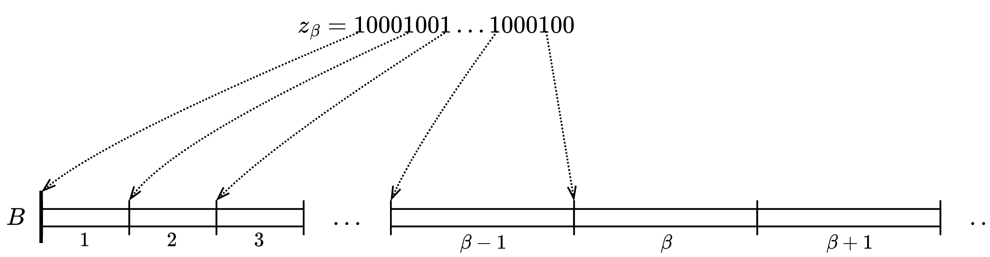

Fig. 5. Example of a string z_β over an array B. The array B is illustrated as being separated into small blocks. The small blocks 1, 2, 3 and $\beta-1$, β, $\beta+1$ are displatyed. The bits with value 1 of string z_β correspond to the left border of some small block, e.g. the first bit with value 1 corresponds to the left border of the small block 1, and the last bit with value 1 corresponds to the left border of the block β.

Definition 4. *For each small block $\beta \in \{1, \ldots, b\}$, having the right endpoint at position r_β of the array B, let H_β be the set of elements $x \in B$, s.t. $\boldsymbol{freq}_x(1, r_\beta) > \beta$.*

We must note that, $|H_\beta| < r_\beta/\beta$, thus, $|H_\beta| = O(\sqrt{r_\beta/w})$.

Further, for each element x, which is present in at least one set H_β, we will define the string η_x the following way:

Definition 5. *For each element x present in at least one set H_β, let β_{max} be rightmost block, s.t. $x \in H_{\beta_{max}}$. Let η_x be a binary string, obeying the following properties:*

Property 7. There are exactly β_{max} bits with value 1 in string η_x, k-th bit with value 1 corresponding to the right border of the k-th small block.

Property 8. For each $k \in \{1, \ldots, \beta_{max}\}$, let r_k be the position of the right endpoint of the k-th small block in array B. There are exactly $\boldsymbol{freq}_x(1, r_k)$ bits with value 0 to the left of the k-th bit with value 1 of string η_x.

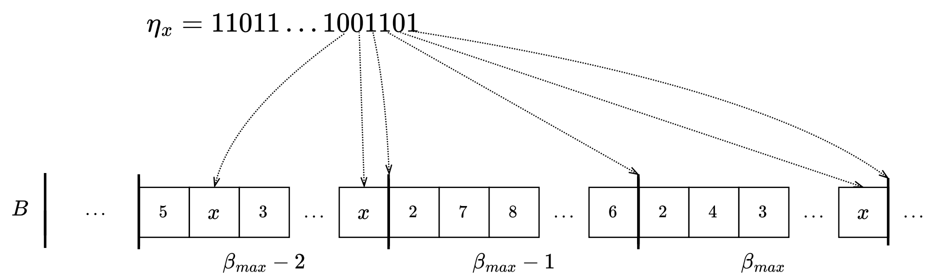

Fig. 6. Example of a string η_x over an array B. The array B is depicted as being separated into small blocks. The small blocks $\beta_{max} - 2$, $\beta_{max} - 1$, β_{max} are depicted, considering that β_{max} is the right most small block s.t. $x \in H_{\beta_{max}}$. For string η_x, each bit with value 0 corresponds to an occurence of x, and each bit with value 1 corresponds to the right border of some small block.

For each block β, we can store each set H_β as an ordered array of pairs of integers (x, p_x), x representing the element of H_β which is being stored, and p_x is a pointer to the data structure storing the string η_x.

As, for each small block β, $|z_\beta| = O(\sqrt{n \cdot w})$ and $b = O(\sqrt{n \cdot w})$ the total space needed to store the strings z_β will be:

$$\sum_{\beta=1}^{b} |z_\beta| = O(\sqrt{n \cdot w}) \cdot O(\sqrt{n \cdot w}) = O(n \cdot w) \text{ bits}$$

$O(n \cdot w)$ bits will be needed to store the strings z_β, thus, $O(n)$ words of RAM will be stored.

As for each small block β, $|H_\beta| = O(\sqrt{n/w})$, the space needed to store the sets H_β will be:

$$\sum_{i=1}^{b} |H_i| = O(\sqrt{n/w}) \cdot O(\sqrt{n \cdot w}) = O(n) \text{ words of RAM.}$$

For each string η_x, let $\beta_{\max}$ be the rightmost small block that contains x, let the number of bits with value 0, be $size_0(\eta_x)$, and the number of bits with value 1 be $size_1(\eta_x)$.

$$size_0(\eta_x) > \beta_{\max}, size_1(\eta_x) = \beta_{\max} \implies size_0(\eta_x) > size_1(\eta_x)$$

Thus, the space required to store the strings η_x will be:

$$\sum_{x\in(\cup_{\beta=1}^{b} H_\beta)} |\eta_x| = \sum_{x\in(\cup_{\beta=1}^{b} H_\beta)} size_0(\eta_x) + size_1(\eta_x)$$
$$\le \sum_{x\in(\cup_{\beta=1}^{b} H_\beta)} 2 \cdot size_0 \eta_x \le \sum_{x\in(\cup_{\beta=1}^{b} H_\beta)} \mathbf{freq}_x(1,n)$$
$$= O(n) \text{ bits of space.}$$

Thus, for maintaining the strings η_x, we need $O(n/w) = o(n)$ words of RAM.
Finally, we may state that the total space required by this data structure is $O(n)$.

Query Algorithm. The query algorithm is as follows:

Given the interval $[i, j]$:

1. We determine the rightmost big block β', of size β', which intersects $[i, j]$. As the length of the prefix of B, covered with big blocks 1 through β', is :

$$\sum_{k=1}^{\beta'} k = \frac{\beta' \cdot (\beta' + 1)}{2} \le 2 \cdot j$$

We can obtain:

$$\beta'^2 \le 2 \cdot (\frac{\beta' \cdot (\beta' + 1)}{2}) \le 4 \cdot j \implies \beta' = O(\sqrt{j})$$

Afterwards, we determine the leftmost and the rightmost small blocks, β_1 and β_2, respectively, that are fully contained in the interval $[i, j]$. Note that:

$$size(\beta_1) \le size(\beta_2)$$

$$size(\beta_2) = \beta'/\sqrt{w} = O(\sqrt{j/w})$$

If there are no small blocks that are fully contained in the interval $[i, j]$, then we can just iterate through all the elements of $[i, j]$ and determine the range mode, which will take $O(\sqrt{j/w})$ time.

2. Let l_1 be the left endpoint of the small block β_1 and r_2 be the right endpoint of the small block β_2 in array B. We query the string z_{β_2}, and get the values $p_2 = |z_{\beta_2}|$, and $p_1 = \mathbf{select}_1(z_{\beta_2}, \beta_1)$ corresponding to the left endpoint of the small block β_1. Further, we calculate:

$$f = (p_2 - p_1) - (\beta_2 - \beta_1)$$

which corresponds to $f = \min(\phi(l_1, r_2), \beta_2)$.
In order to get an element x, which has frequency at least f in the interval of small blocks $[\beta_1, \beta_2]$, we can get the rightmost position p_f, corresponding to the left end of the small block β_f, such that, there are exactly f bits with

value 0 in the interval $[p_f, p_2]$ in the string η_x. This can be done easily, using **rank** and **select** operations, as stated by Chan et al.[1]. Then, we need to iterate through each position k of the small block β_f, and check whether there are at least f occurrences of B_k inside the interval of small blocks $[k, r_2]$.

3. Further, we iterate through the elements x of H_{β_2+1}, and find the values of their frequencies in the interval of small blocks $[\beta_1, \beta_2]$. In order to find the frequency of element x in the interval of small blocks $[\beta_1, \beta_2]$, we query the string η_x for values:

$$p_1 = \mathbf{select}_1(\eta_x, \beta_1 - 1)$$

$$p_2 = \mathbf{select}_1(\eta_x, \beta_2)$$

   p_1 being the position of the bit with value 1 in string η_x, corresponding to the right endpoint of the small interval $\beta_1 - 1$, and p_2, being the position of the bit with value 1, corresponding to the right endpoint of the small interval β_2. Thus, the frequency of the element x will be:

$$f_x = (p_2 - p_1) - (\beta_2 - \beta_1 + 1)$$

   Further, we update the value of f the following way:

$$f := \max(f, \max_{x \in H_{\beta_2+1}} (f_x))$$

   This way, in a manner similar to that presented in **lemma 1** and **lemma 2**, we obtain the value $\phi(l_1, r_2)$, which is exactly the frequency of a mode, in the interval corresponding to the small blocks $\{\beta_1, \beta_1 + 1, \ldots, \beta_2\}$. We must note, that during this step, it is also easy to keep track of an element x of maximum frequency.

4. Now, we must iterate through positions $k \in ([i, j] \setminus [l_1, r_2])$ and try to increase the frequency f, using the arrays Q, B and Q^{-1}, as in **lemma 1** and **lemma 2**. This will take at most $O(\sqrt{j/w})$ time, which is an upper bound on the sizes of the small blocks β_1 and β_2.

Analysis. This query algorithm will clearly require $O(\sqrt{j/w})$ time, as step 2 takes at most $O(\sqrt{j/w})$ time, step 3 takes at most $O(\sqrt{r_2/w}) = O(\sqrt{j/w})$ time, and step 4 similarly, takes at most $O(\sqrt{j/w})$ time. □

Algorithm 3 Lemma 3 Query Algorithm

function QUERY

▷ Step 1:

  $\beta' \leftarrow \lceil -\frac{1}{2} + \sqrt{\frac{1}{4} + 2 \cdot j} \rceil$

  $\beta_2 \leftarrow \lfloor \frac{(j - \frac{\beta'(\beta'-1)}{2})}{\frac{\beta'}{\sqrt{w}}} \rfloor + (\beta' - 1) \cdot \sqrt{w}$

  $t \leftarrow \lceil -\frac{1}{2} + \sqrt{\frac{1}{4} + 2 \cdot i} \rceil$

  $v \leftarrow \lfloor \frac{(i - \frac{t(t-1)}{2})}{\frac{t}{\sqrt{w}}} \rfloor + (t - 1) \cdot \sqrt{w}$

  $\beta_1 \leftarrow v + 2$

  if $(v - (t-1) \cdot \sqrt{w}) \cdot \frac{t}{\sqrt{w}} + \frac{t \cdot (t-1)}{2} + 1 = i$ **then**

    $\beta_1 \leftarrow v + 1$

  if $\beta_1 > \beta_2$ **then**

    #Iterate through each element of [i,j] and return the answer.

▷ Step 2:

  $l_1 \leftarrow (\beta_1 - 1 - (t-1) \cdot \sqrt{w}) \cdot \frac{t}{\sqrt{w}} + \frac{t \cdot (t-1)}{2} + 1$

  $r_2 \leftarrow (\beta_2 - (\beta' - 1) \cdot \sqrt{w}) \cdot \frac{\beta'}{\sqrt{w}} + \frac{\beta' \cdot (\beta'-1)}{2}$

  $p_2 \leftarrow |z_{\beta_2}|, p_1 \leftarrow \mathbf{select}_1(z_{\beta_2}, \beta_1)$

  $f \leftarrow f = (p_2 - p_1) - (\beta_2 - \beta_1)$

  # Find β_f and p_f, and iterate through the block β_f, and get the candidate element m.

▷ Step 3:

  for $x \in H_{\beta_2+1}$ **do**

    $p_1^x = \mathbf{select}_1(\eta_x, \beta_1 - 1)$

    $p_2^x = \mathbf{select}_1(\eta_x, \beta_2)$

    $f_x \leftarrow (p_2^x - p_1^x) - (\beta_2 - \beta_1 + 1)$

    if $f_x > f$ **then**

      $f \leftarrow f_x$

      $m \leftarrow x$

▷ Step 4:

  for $k \in [i, l_1 - 1]$ **do**

    while $Q_{B_k}(Q^{-1}(k) + f + 1) \leq j$ **do**

      $f \leftarrow f + 1$

      $m \leftarrow B_k$

  for $k \in [r_2 + 1, j]$ **do**

    while $Q_{B_k}(Q^{-1}(k) - f - 1) \geq i$ **do**

      $f \leftarrow f + 1$

      $m \leftarrow B_k$

  return $(C(m), f)$

Theorem 2. *Given an array $A[1 : n]$, there exists a data structure, requiring $O(n)$ words of RAM, that supports range mode queries over an interval $[i, j]$ in time $O(\min(\sqrt{j - i + 1}, \sqrt{j/w}))$.*

Proof. Firstly, we will build the arrays B, C, Q, Q^{-1} and the set D, which will take $O(n \log n)$ time and $O(n)$ space. Further, we will build an instance of the data structure defined in **lemma 3**, denoted by DS_1, and an instance of the data structure defined in **theorem 1**, denoted by DS_2. The arrays, as well as both of the instances of the data structures mentioned above, will require $O(n)$ space, thus, the space required by the current data structure is $O(n)$.

Query Algorithm. The query algorithm is as follows:

Given the interval $[i, j]$:
1. Estimate the time required by a query over the data structure DS_1 and the interval $[i, j]$. Let this time estimation be t_1. Estimate the time required by a query over the data structure DS_2 and the interval $[i, j]$. Let this time estimation be t_2.
2. If $t_1 < t_2$, then, return $DS_1.query(i, j)$, otherwise, return $DS_2.query(i, j)$.

Analysis. It is easy to do the time estimates t_1 and t_2 up to a constant factor, in at most logarithmic time.

Thus, the runtime of the query algorithm is $O(\min(\sqrt{j-i+1}, \sqrt{j/w}))$.□

3 Adding Elements at the End of the Array

Further, considering the fact, that we can implement a compact, data structure, supporting rank/select operations in $O(1)$ time, and adding a bit to the end of the array in amortyzed $O(1)$ time, we will show how to make the data structure from section 2 support adding an element at the end of the array $A[1:n]$ in amortyzed $O(\sqrt{n \cdot w})$ time, while requiring linear space and keeping the same query time.

Lemma 4. *Given an array $A[1:n]$, there exists a data structure, that respects the same conditions as the data structure constructed in Lemma 1, and also supports adding an element at the end of the array $A[1:n]$, in amortyzed $O(\sqrt{2^L})$ time.*

Proof. As the only data stored in the data structure from **lemma 1**, are the strings $s^L_{\beta_1}$, we will present an algorithm to update $s^L_{\beta_1}$, as new elements are added to the end of array A.

Update Algorithm. Suppose that there are β' big blocks of size exactly $b'_L = 2^L$ that are fully contained in array $B[1:n]$. Let β be the number of small blocks, of size $b_L = \sqrt{2^L}$, that are fully contained in the array $B[1:n]$. Let $\beta_{new} = \beta + 1$ be the new small block, which is constructed as new elements are appended to the end of $A[1:n]$.

An update is given, in the form of a number x, that must be added to the end of $A[1:n]$.

1. Determine the element y of array B, s.t. $C_y = x$, then add $n+1$ to the end of Q_y, and add $|Q_y|$ to the end of Q^{-1}. If such an element y does not exist, then, $\Delta := \Delta + 1$ and $y := \Delta$, add x to the end of the array C, define $Q_y = \{n+1\}$, and add 1 to the end of Q^{-1}. Add y to the end of B.
2. For each small block $\beta_i \in \{\beta_{new} - \sqrt{2^L} + 1, \ldots, \beta_{new}\}$, let l_i be the left endpoint of β_i. We must check, whether the new element y added to $B[1:n]$, changes the maximum frequency inside the interval of small blocks $[\beta_i, \beta_{new}]$, by verifying the following condition:

$$p = |s^L_{\beta_i}|, c_1 = \mathbf{rank}_1(s^L_{\beta_i}, p)$$

$$f = p - c_1, l_y = Q_y(Q^{-1}(n+1) - f - 1)$$

$$l_y \geq l_i \text{ and } f + 1 \leq \sqrt{2^L}$$

   If this condition holds, then we add a bit with value 0 to the end of $s^L_{\beta_i}$. This will signify the increase in the maximum frequency of an element, recorded by the string $s^L_{\beta_i}$. We must also note, that the strings $s^L_{\beta_i}$ will change their structure during these intermediary frequency-increase steps, as bits with value 0 will be present at the end of $s^L_{\beta_i}$. This will not influence the queries over $s^L_{\beta_i}$ required by the data structure from **lemma 1**.
3. If the position $n+1$ is the right endpoint of a new small block β_{new} that is now fully contained in array $B[1:(n+1)]$, then, for each $\beta_i \in \{\beta_{new} - \sqrt{2^L} + 1, \ldots, \beta_{new}\}$, we must update the information stored in $s^L_{\beta_i}$. In such a case, we must add 1 to the end of $s^L_{\beta_i}$, which will signify the addition of a right endpoint of a new full small block.
4. Finally, we add element x to the end of array A, and set $n := n+1$.

Analysis. It is easy to see that the first step takes at most logarithmic time. The second step, will take amortyzed $O(\sqrt{2^L})$ time, as there may be at most $O(\sqrt{2^L})$ small blocks β_i, and each append of a bit to the end of β_i takes amortyzed $O(1)$ time. Similarly, the third step takes $O(\sqrt{2^L})$ time.

The required space will still remain linear, as $s^L_{\beta_1}$ has been proven to require $O(n/w)$ words of RAM, and $O(1)$ more bits of space will be added at each update. □

Algorithm 4 Lemma 4 Update Algorithm

function UPDATE
▷ Step 1:
  $y \leftarrow 0$
  if $x \notin D$ **then**
    $\Delta \leftarrow \Delta + 1$
    $y \leftarrow \Delta$
    $D.insert(x)$
    $Q_y \leftarrow \{n+1\}$
    $Q^{-1}.append(1)$
  else
    $y \leftarrow D.rank(x)$
    $Q_y.append(n+1)$
    $Q^{-1}.append(|Q_y|)$
  $B.append(y)$
▷ Step 2:
  for $\beta_i \in \{\beta_{new} - \sqrt{2^L} + 1, \ldots, \beta_{new}\}$ **do**
    $p = |s^L_{\beta_i}|, c_1 = \mathbf{rank}_1(s^L_{\beta_i}, p)$
    $f = p - c_1, l_y = Q_y(Q^{-1}(n+1) - f - 1)$
    if $l_y \geq l_i$ and $f + 1 \leq \sqrt{2^L}$ **then**
      $s^L_{\beta_i}.append(0)$
▷ Step 3:
  if i is the right endpoint of β_{new} **then**
    for $\beta_i \in \{\beta_{new} - \sqrt{2^L} + 1, \ldots, \beta_{new}\}$ **do**
      $s^L_{\beta_i}.append(1)$
▷ Step 4:
  $A.append(x)$
  $n \leftarrow n + 1$

Lemma 5. *Given an array $A[1:n]$, there exists a data structure, that respects the same conditions as the data structure constructed in Lemma 2, and also supports adding an element at the end of the array $A[1:n]$, in amortyzed $O(\sqrt{2^L})$ time.*

Proof. As the only additional data stored in the data structure from **lemma 2**, are the sets F^L_i and the strings $Y^L_{i,x}$, we will present an algorithm to update them, as new elements are added to the end of array A.

Update Algorithm. Suppose that there are β' big blocks of size exactly $b'_L = 2^L$ that are fully contained in array $B[1:n]$. Let β be the number of small blocks, of size $b_L = \sqrt{2^L}$, that are fully contained in the array $B[1:n]$. Let $\beta_{new} = \beta + 1$ be the new small block, which is constructed as new elements are appended to the end of $A[1:n]$, and $\beta'_{new} = \beta' + 1$ be the new big block, which is constructed as new elements are appended to the end of $A[1:n]$.

An update is given, in the form of a number x, that must be added to the end of $A[1:n]$.

1. Determine the element y of array B, s.t. $C_y = x$, then add $n+1$ to the end of Q_y, and add $size(Q_y)$ to the end of Q^{-1}. If such an element y does not exist, then, $\Delta := \Delta + 1$ and $y := \Delta$, add x to the end of the array C, define $Q_y = \{n+1\}$, and add 1 to the end of Q^{-1}. Add y to the end of B.
2. We must first check, if the frequency of the element y in the interval of B, corresponding to the interval of big blocks $[\beta', \beta'_{new}]$ is $> \sqrt{2^L}$. According to Chan et al.[1], this can be done in $O(\log n)$ time, using the arrays Q_y and Q^{-1}.
   If the frequency of y is not big enough, then we can stop the update procedure.
   Otherwise, we must check whether the element y is present in $F^L_{\beta'}$. This can be done simply, by iterating through the elements of $F^L_{\beta'}$.
   If y is present in $F^L_{\beta'}$, then we must add 0 to the end of $Y^L_{\beta',y}$. Otherwise, we must create the string $Y^L_{\beta',y}$, and add the pair (y, p_y) to the end of $F^L_{\beta'}$.
3. In this step, we will present a procedure, to create the string $Y^L_{\beta',y}$ when y has to be added to the set $F^L_{\beta'}$ for the first time.
   (a) Create the string $Y^L_{\beta',y}$, and the pointer p_y.
   (b) Let p be the leftmost position, s.t. $B_p = y$, in the interval of B, corresponding to the interval of big blocks $[\beta', \beta'_{new}]$. Let β_p be the small blocks that contains the position p. These two values can be found trivially in $O(\sqrt{2^L})$ time.
   Let β_{first}, be the leftmost small block, that is contained in the big block β'. We insert $\beta_p - \beta_{first}$ bits with value 1 to the end of $Y^L_{\beta',y}$. Note that $\beta_p - \beta_{first} = O(\sqrt{2^L})$.
   (c) Insert a bit with value 0 to the end of $Y^L_{\beta',y}$. If $p = n+1$, it means that we have evaluated currently the last position at which the value y can be found in B, and we must stop the construction procedure. Set $p := Q_y(Q^{-1}(p)+1)$. If p is outside of the small block β_p, then, set $\beta_p := \beta_p + 1$, and insert a bit with value 1 to the end of $Y^L_{\beta',y}$.
   (d) Go to step (c).
4. In a manner similar to steps 2 and 3, we update, or create if not found, the string $Y^L_{\beta'_{new},y}$ and the set $F^L_{\beta'_{new}}$.
5. Further, we check whether the current position is the rightmost position contained in the big block β'. If this is true, and the string $Y^L_{\beta',y}$ exists, then we add a bit with value 1 to the end of $Y^L_{\beta',y}$. In the same way, if $n+1$ is the rightmost position of the big block β'_{new} and the string $Y^L_{\beta'_{new},y}$ exists, then we add a bit with value 1 to the end of $Y^L_{\beta'_{new},y}$.
6. As a final step, we append element x to the end of array A, and set $n := n+1$.

Analysis. It is easy to see that the first step takes at most logarithmic time. The second step takes at most $O(\sqrt{2^L})$ amortyzed time. The procedure described in step 3, takes $O(\sqrt{2^L})$ time, as we begin iterating through the positions of y, only the first time, when the frequency of y relative to the big blocks β' and β'_{new} increases beyond $\sqrt{2^L}$. Thus, in step 3, we chekc only $O(\sqrt{2^L})$ positions of y, and at most $O(\sqrt{2^L})$ small blocks, thus, we need only $O(\sqrt{2^L})$ amortyzed time. The 4-th step is similar to the 2-nd step followed by the 3-rd one, thus it takes $O(\sqrt{2^L})$ amortyzed time. The last step takes amortyzed $O(1)$ time.

Thus, the update algorithm runs in $O(\sqrt{2^L})$ amortyzed time.□

Algorithm 5 Lemma 5 Update Algorithm

function UPDATE
▷ Step 1:
  $y \leftarrow 0$
  if $x \notin D$ **then**
    $\Delta \leftarrow \Delta + 1$
    $y \leftarrow \Delta$
    $D.insert(x)$
    $Q_y \leftarrow \{n+1\}$
    $Q^{-1}.append(1)$
  else
    $y \leftarrow D.rank(x)$
    $Q_y.append(n+1)$
    $Q^{-1}.append(|Q_y|)$
  $B.append(y)$
▷ Step 2,3:
  if the frequency of y is $> \sqrt{2^L}$ in the interval of big blocks $[\beta', \beta'_{new}]$ **then**
    if $y \in F^L_{\beta'}$ **then**
      $Y^L_{\beta',y}.append(0)$
    else
      $(Y^L_{\beta',y}, p_y) \leftarrow buildY(\beta', y, L)$
      $F^L_{\beta'}.insert(y)$
▷ Step 4:
  if the frequency of y is $> \sqrt{2^L}$ in the interval of big block β'_{new} **then**
    if $y \in F^L_{\beta'_{new}}$ **then**
      $Y^L_{\beta'_{new},y}.append(0)$
    else
      $(Y^L_{\beta'_{new},y}, p_y) \leftarrow buildY(\beta'_{new}, y, L)$
      $F^L_{\beta'_{new}}.insert(y)$
▷ Step 5:
  if $n+1$ is the right endpoint of the big block β'_{new} **then**
    if $Y^L_{\beta',y}$ exists **then**
      $Y^L_{\beta',y}.append(1)$
    if $Y^L_{\beta'_{new},y}$ exists **then**
      $Y^L_{\beta'_{new},y}.append(1)$
▷ Step 6:
  $A.append(x)$
  $n \leftarrow n+1$

Algorithm 6 Lemma 5 Algorithm for building $Y^L_{\beta',y}$

function BUILDY(β', y, L)
    Create $Y^L_{\beta',y}$ and the pointer p_y
    Let p be the leftmost position in B of the element y, in the interval of blocks $[\beta', \beta'_{new}]$
    $\beta_p \leftarrow \lfloor (p + \sqrt{2^L})/\sqrt{2^L} \rfloor$
    $\beta_{first} \leftarrow (\beta' - 1) \cdot \sqrt{2^L} + 1$
    for $k \in [1, \beta_p - \beta_{first}]$ **do**
        $Y^L_{\beta',y}.append(1)$
    while $p < n + 1$ **do**
        $Y^L_{\beta',y}.append(0)$
        $p \leftarrow Q_y(Q^{-1}(p) + 1)$
        if p is outside the small block β_p **then**
            $\beta_p \leftarrow \beta_p + 1$
            $Y^L_{\beta',y}.append(1)$
    return $(Y^L_{\beta',y}, p_y)$

Further, we will summarize the results from **lemma 4** and **lemma 5** and show how to achieve amortyzed $O(w + \sqrt{n})$ runtime for updates, while keeping the other properties for the data structures described in **lemma 1** and **lemma 2**.

Theorem 3. *Given an array $A[1:n]$, there exists a data structure, requiring arrays B, C, Q, and additional $O(n)$ words of RAM, that supports range mode queries over an interval $[i,j]$ in $O(\sqrt{j-i+1})$ time, and supports adding an element at the end of $A[1:n]$ in amortyzed $O(w + \sqrt{n})$ time.*

Proof. As declared in **lemma 4** and **lemma 5**, the data structures from **lemma 1** and **lemma 2** can be modified in order to support updates in $O(\sqrt{2^L})$ time, while requiring the same amoung of space and maintaining a query in $O(\sqrt{2^L})$ time.

Further, for each $L \in \{0, \ldots, \lceil \log n \rceil\}$, we will build an instance of the data structure described in **lemma 4**, denoted by DS^L_1, and an instance of the data structure described in **lemma 5**, denoted by DS^L_2. These instances will take $O(n)$ space, as the arrays B, C, Q, Q^{-1}, F^L_i and the set D, will be shared among them, and the additional information will take $O(\log n \cdot n/w) = O(n)$ space.

As the query time remains unchanged, we must study only the update time. As for one update, at a moment in time when the array A has length n, for each $L \in \{0, \ldots, \lceil \log n \rceil\}$ the update time if $O(\sqrt{2^L})$, the total update time for all $L \in \{0, \ldots, \lceil \log n \rceil\}$, DS^L_1 and DS^L_2 will be:

$$\sum_{L=0}^{\lceil \log n \rceil} \lceil \sqrt{2^L} \rceil \le w + \sqrt{n} \cdot \sum_{L=0}^{\lceil \log n \rceil} \frac{1}{\sqrt{2^L}} \le \sqrt{n} \cdot \sum_{L=0}^{\inf} \frac{1}{\sqrt{2^L}}$$
$$\le w + O(\sqrt{n}) = O(w + \sqrt{n}) \square$$

Lemma 6. *Given an array $A[1:n]$, there exists a data structure requiring $O(n)$ words of RAM, that supports range mode queries over an interval $[i,j]$ in $O(\sqrt{j/w})$ time, and supports adding an element at the end of $A[1:n]$ in amortyzed $O(\sqrt{n \cdot w})$ time.*

Proof. As the only additional data stored in the data structure from **lemma 3**, are the sets H_β and the strings z_β and η_x, we will present an algorithm to update them, as new elements are added to the end of array A.

Firstly, we suppose there are β' big blocks, fully contained in array $B[1:n]$, and β be the number of small blocks fully contained in array $B[1:n]$. Let $\beta_{new} = \beta + 1$ be the new small block, constructed while new elements are appended to A, and, similarly, let $\beta'_{new} = \beta' + 1$ be the new big block constructed while new elements are appended to A.

In our approach, we also introduce the array τ of size β_{new}, defined below:

Definition 6. *Let array τ, be an array of integers, of size β_{new}, obeying the following properties:*

Property 9. For each small block $k \in \{1, \ldots, \beta_{new}\}$, having l_k as its left endpoint:

$$\tau_k = \phi(l_k, n)$$

In other words, in τ, we will store a decreasing array, describing the frequency of the most frequent element in the last small blocks of the array B.

Update Algorithm.

An update is given, in the form of a number x, that must be added to the end of A[1 : n].

1. Determine the element y of array B, s.t. $C_y = x$, then add $n+1$ to the end of Q_y, and add $size(Q_y)$ to the end of Q^{-1}. If such an element y does not exist, then, $\Delta := \Delta + 1$ and $y := \Delta$, add x to the end of the array C, define $Q_y = \{n+1\}$, and add 1 to the end of Q^{-1}. Add y to the end of B.
2. In this step, we will iterate through each position k in τ, and check whether we can increase τ_k. We will use the fact, that the addition of the element y to the end of B, will increase each value stored in τ by at most 1, because, for each k, $\phi(l_k, n+1) \leq 1 + \phi(l_k, n)$.
   This procedure is summarized in the following sub-steps:
   (a) Let $\beta_{current} := \beta_{new}$.
   (b) Let $l_{\beta_{current}}$ be the left endpoint of the small block $\beta_{current}$.
   (c) If $Q_y(|Q_y| - \tau_{\beta_{current}}) \geq l_k$, then set $\tau_{\beta_{current}} := \tau_{\beta_{current}} + 1$.
   (d) If $\beta_{current} > 1$, set $\beta_{current} := \beta_{current} - 1$, and go back to step (b).
   This way, we update the values of τ, in $O(\beta_{new})$ time.
3. If $y \in H_{\beta_{new}}$, then, we add a bit with value 0 to the end of η_y. Otherwise, we check if $\mathbf{freq}_y(1, n) > \beta_{new}$. If this is true, then we will add y to $H_{\beta_{new}}$, and update the string η_y. If η_y did not exist, we will suppose that it was empty. We descrive the process of updating the string η_y, by the following procedure:

(a) Let $\beta_{current} := \mathbf{rank}_1(\eta_y, |\eta_y|)$, $p := \mathbf{rank}_0(\eta_y, |\eta_y|) + 1$, where $\beta_{current}$ will represent the current small block, and p will represent the position in array Q_y, of the first position of y that has not been considered in the string η_y yet.
(b) Let $r_{current}$ be the right endpoint of the small block $\beta_{current}$.
(c) If $p \le r_{current}$, then, set $p := p + 1$, add 0 to η_y, and return to step $3.b$.
(d) If $\beta_{current} < \beta_{new}$, then set $\beta_{current} := \beta_{current} + 1$, add 1 to the end of η_y and go back to step $3.b$.

4. If the position $n + 1$ is the right endpoint of the small block β_{new}, then we iterate through each $y \in H_{\beta_{new}}$ and add 1 to the end of η_y.
5. If the position $n + 1$ is the right endpoint of the small block β_{new}, then, for each $y \in H_{\beta_{new}}$, if it has frequency $\ge \beta_{new} + 1$, add y to $H_{\beta_{new}+1}$.
6. If the position $n + 1$ is the right endpoint of the small block β_{new}, then we have to build $z_{\beta_{new}}$, by using the array τ.
   We will describe the process of constructing $z_{\beta_{new}}$, by the following procedure:
   (a) Let $k := \beta_{new}$, and $f := 0$.
   (b) If $f < \min(\tau_k, \beta_{new})$, then, set $f := f + 1$, add 0 to the end of $z_{\beta_{new}}$ and return to step $5.b$.
   (c) If $k > 0$, add 1 to the end of $z_{\beta_{new}}$, set $k := k - 1$ and go to step $5.b$.
   (d) Now, we will reverse the string $z_{\beta_{new}}$, in order to make it obey the definition declared in **lemma 3**.

   Now, we will set $\beta_{new} := \beta_{new} + 1$, s.t. β_{new} will now have the value of the small block, that will begin its construction when an element will be append to B at position $n + 2$. Also, we will append a new integer with value 0 to the end of τ.
7. Finally, we append x to A, and set $n := n + 1$.

Analysis. It is easy to see that the first step takes at most logarithmic time. If the position $n+1$ is the right endpoint of the small block The second step will take $O(\beta_{new}) = O(|\tau|) = O(\sqrt{n \cdot w})$ time.

The third step will take at most $O(\beta_{new}) = O(\sqrt{n \cdot w})$ time, because, if y was already in $H_{\beta_{new}}$, then we need only $O(1)$ time to update η_y, and if y was not in $H_{\beta_{new}}$ yet, then the frequency of y currently is at most $\beta_{new} + 1$, and the construction of η_y will take at most $O(\beta_{new}) = O(\sqrt{n \cdot w})$ time.

The fourth step will take at most $O(\sqrt{n/w})$ time, as well as the fifth step, as we will just need to iterate through each element $y \in H_{\beta_{new}}$, and $|H_{\beta_{new}}| = O(\sqrt{n/w})$.

The sixth step will take $O(\sqrt{n \cdot w})$ time, as we will iterate through the possible values of f from 0 to at most β_{new}, and through each entry of τ, and $O(\beta_{new} + |\tau|) = O(\beta_{new}) = O(\sqrt{n \cdot w})$.

The seventh step, will take $O(1)$ time. Thus, the update algorithm will require $O(\sqrt{n \cdot w})$ time, and will keep the linear-space restriction for our data structure.□

Algorithm 7 Lemma 6 Update Algorithm

function UPDATE
▷ Step 1:
    $y \leftarrow 0$
    if $x \notin D$ **then**
        $\Delta \leftarrow \Delta + 1$
        $y \leftarrow \Delta$
        $D.insert(x)$
        $Q_y \leftarrow \{n+1\}$
        $Q^{-1}.append(1)$
    else
        $y \leftarrow D.rank(x)$
        $Q_y.append(n+1)$
        $Q^{-1}.append(|Q_y|)$
    $B.append(y)$
▷ Step 2:
    for $\beta_{current} \in \{1, \ldots, \beta_{new}\}$ **do**
        Let $l_{\beta_{current}}$ be the left endpoint of the small block $\beta_{current}$
        if $Q_y(Q^{-1}(n+1) - \tau_{\beta_{current}} - 1) \geq l_{\beta_{current}}$ **then**
            $\tau_{\beta_{current}} \leftarrow \tau_{\beta_{current}} + 1$
▷ Step 3:
    if $\mathbf{freq}_y(1, n) > \beta_{new}$ **then**
        if $y \in H_{\beta_{new}}$ **then**
            $\eta_y.append(0)$
        else
            $\eta_y \leftarrow update\_\eta(y)$
            $H_{\beta_{new}}.insert(y)$
▷ Steps 4,5,6:
    if $n + 1$ is the right endpoint of the small block β_{new} **then**
        for $y \in H_{\beta_{new}}$ **do**
            $\eta_y.append(1)$
            if $\mathbf{freq}_y(1, n) > \beta_{new} + 1$ **then**
                $H_{\beta_{new}+1}.insert(y)$
        $z_{\beta_{new}} \leftarrow build\_z(\beta_{new}, y)$
        $\beta_{new} \leftarrow \beta_{new} + 1$
        $\tau.append(0)$
▷ Step 7:
    $A.append(x)$
    $n \leftarrow n + 1$

Algorithm 8 Lemma 6 Algorithm for updating/building η_y

function UPDATE_$\eta(y)$
  if η_y does not exist **then**
    Create the string η_y
  $\beta_{current} \leftarrow \mathbf{rank}_1(\eta_y, |\eta_y|)$
  $p \leftarrow \mathbf{rank}_0(\eta_y, |\eta_y|) + 1$
  while $\beta_{current} < \beta_{new}$ or $(\beta_{current} = \beta_{new} p \leq r_{current})$ **do**
    Let $r_{current}$ be the right endpoint of the small block $\beta_{current}$
    if $p < r_{current}$ **then**
      $p \leftarrow p + 1$
      $\eta_y.append(0)$
    $\beta_{current} \leftarrow \beta_{current} + 1$
    $\eta_y.append(1)$
  return η_y

Algorithm 9 Lemma 6 Algorithm for building $z_{\beta_{new}}$

function BUILD_Z(β_{new})
  if $z_{\beta_{new}}$ does not exist **then**
    Create the string $z_{\beta_{new}}$
  $f \leftarrow 0$
  for $k \leftarrow \beta_{new}$ **downto** 1 **do**
    while $f < \min(\tau_k, \beta_{new})$ **do**
      $f \leftarrow f + 1$
      $z_{\beta_{new}}.append(0)$
    $z_{\beta_{new}}.append(1)$
  $z_{\beta_{new}} \leftarrow reverse(z_{\beta_{new}})$
  return $z_{\beta_{new}}$

Theorem 4. *Given an array $A[1:n]$, there exists a data structure, requiring $O(n)$ words of RAM, that supports range mode queries over an interval $[i,j]$ in $O(\min(\sqrt{j-i+1}, \sqrt{j/w}))$ time, and supports adding an element at the end of $A[1:n]$ in amortyzed $O(w + \sqrt{n \cdot w})$ time.*

Proof. Firstly, we will build the arrays B, C, Q, Q^{-1} and the set D, which will take $O(n \log n)$ time and $O(n)$ space. Further, we will build an instance of the data structure defined in **lemma 6**, denoted by DS_1, and an instance of the data structure defined in **theorem 3**, denoted by DS_2. The arrays, as well as both of the instances of the data structures mentioned above, will require $O(n)$ space, thus, the space required by the current data structure is $O(n)$.

Note that the time required by the query algorithm will remain the same as the runtime of the data structure defined in **theorem 2**, more exactly:
$O(\min(\sqrt{j-i+1}, \sqrt{j/w}))$.

The update algorithm will consist only of calling the methods $DS_1.update(x)$ and $DS_2.update(x)$, given an integer x that will be appended to the end of A. Thus, the time required by the update algorithm for this data structure will be:

$$O(w + \sqrt{n}) + O(\sqrt{n \cdot w}) = O(w + \sqrt{n \cdot w})\square$$

4 Compact rank/select data structure

We present a compact data structure, that supports $O(1)$ rank/select/access operations over a binary array B of length Z, by using $O(Z)$ additional bits of space, and supports adding bits, one-by-one, at the end of array B in amortyzed $O(1)$ time.

In the construction of our data structures, we will use the fact that we can implement an array for storing integers representable in a single word of RAM, that supports **access** operations in constant $O(1)$ time, and supports appending an integer to the end of the array in amortyzed $O(1)$ time, while requiring linear space.

Firstly, we will consider the following result:

Lemma 7. *There exists a data structure, that can store a binary array B of size Z, by using $O(Z)$ bits, and will support **rank** and **access** operations in $O(1)$ time, and appending a bit to the end of the array in amortyzed $O(1)$ time.*

Proof. Consider $w' = \lceil \frac{\log Z}{2} \rceil$. We will separate the array B into adjacent blocks of size w', and will represent each block, as a number stored as a word of RAM of size w. The array B, will be represented as an array of numbers B', B'_i containing the bits of the block i.

We will build the array P, with size equal to that of B', where each entry P_i will be equal to the sum of bits with value 1, in the blocks 1 through i.

We will build a look-up table T, of size $O(2^{w'} \cdot w') = O(\sqrt{Z} \cdot \log Z)$words of RAM, s.t.:

$$T_x(i) = |\{j | 0 \le j \le i, \text{ the } j\text{-th bit } x \text{ has value } 1\}|$$

Note that we consider the bits of a natural number x to be 0-indexed. We will also consider $T_x(-1) = 0$ for convenience.

Considering the fact that the size of a word of RAM is $w = \Theta(Z)$, the space required by the data structure will be:

$$|B'| \cdot w + |P'| \cdot w + |T| \cdot w = O(Z/w') \cdot w + O(\sqrt{Z} \cdot \log Z) \cdot w = O(Z) \text{ bits.}$$

Rank Query Algorithm.

Given an index i of the array B, we will calculate $\mathbf{rank}_1(B, i)$ the following way:

1. Let $b = \lfloor \frac{i-1}{w'} \rfloor$ be the index in B' of the block immediately to the left of the block that contains the index i.
2. Return $P'(b) + T(B'_{b+1}, i - w' \cdot b - 1)$.

It is easy to see that each step of the rank query algorithm will take $O(1)$ time.

We will also note, that the answer to a query of the form $\mathbf{rank}_0(B, i)$, is $i - \mathbf{rank}_1(B, i)$, which can be easily calculated in $O(1)$ time.

Access Query Algorithm.

Given an index i of the array B, we will get the value of the bit at position i the following way:

1. Let $b = \lfloor \frac{i-1}{w'} \rfloor$ be the index in B' of the block immediately to the left of the block that contains the index i.
2. Return $T_x(B'_{b+1}, i - w' \cdot b - 1) - T_x(B'_{b+1}, i - w' \cdot b - 2)$.

It is easy to see that each step of the access query algorithm will take $O(1)$ time.

Update Algorithm.

Given a bit with value v, the algorithm to append v to the end of B is described the following way:

1. Let $b = \lfloor \frac{Z}{w'} \rfloor$. If $|B'| = b$, then add a new number with all bits set to 0 to the end of B'.
2. We will set the value of the $Z - b \cdot w'$ bit of the $b + 1$-th number in B' to v.
3. Set $Z := Z + 1$.

It is easy to see that each step of the update algorithm will take $O(1)$ time.□

Lemma 8. *There exists a data structure, that can store a binary array B of size Z, by using $O(Z)$ bits, and will support* $\mathbf{select}_1$ *operations in $O(1)$ time, and appending a bit to the end of the array in amortyzed $O(1)$ time.*

Proof. Similarly to the data structure from **lemma 7**, we will build the arrays B' and P.

We will keep c_1-the value of bits with value 1 in B.

We will also keep a binary array S, of size at most c_1, through an instance of the data structure from **lemma 7**, denoted by DS_S. Each bit i from S, will have value 1, if there exists an entry in P of value i, and 0 otherwise. If the position Z is the end of a block of size w', the string S will be of length exactly c_1. Otherwise, the length of S will be $c_1 - 1$, and only information about the first $c_1 - 1$ possible values of a prefix sum over B will be stored. Storing S in such a manner, will prove itself useful for the update algorithm.

We will build the array V of positive integers, of size at most $|B'|$, V_k having the value $\mathbf{select}_1(S, k)$.

We will build the arrays L and R, of size $|B'|$. L_k will be the left-most index in array B' of the block b, s.t. $P_b = j$, where j is the position in array S, of the k-th bit with value 1, counting from left to right. In other words, $P_b = \mathbf{select}_1(S, k)$.

Similarly, R_k will be the rightmost index in array B' of the block b, s.t. $P_b = \mathbf{select}_1(S, k)$.

We will build the lookup table T' of size $O(2^{w'} \cdot w') = O(\sqrt{Z} \cdot \log Z)$, s.t.:

$T'_x(i) = j$ s.t., the j-th bit in x, is the i-th bit with value 1 of number x

For convenience, if there are at most m bits with value 1 in x, then, for any value $M > m$, $T'_x(M) = -1$.

Query algorithm.

Given an index i, we will calculate $\mathbf{select}_1(B, i)$ the following way:

1. Let $k = DS_S.\mathbf{rank}_1(i)$;
2. If $i = V_k$, then return $(L_k - 1) \cdot w' + T'_{B'_{L_k}}(i - P_{(L_k-1)} - 1)$;
3. Return $T'_{B'(R_k+1)}(i - V_k - 1) + R_k \cdot w'$;

It is easy to see that each step of the query algorithm will take $O(1)$ time.

Update Algorithm .

Given a bit with value v, we will append it to the end of B the following way:

1. Let $b = \lfloor \frac{Z}{w'} \rfloor$.
2. If $v = 1$ and $T'_{B'_{b+1}}(0) \geq 0$, call $DS_S.update(0)$.
3. Set the $(Z - b \cdot w')$-th bit of B' to value v.
4. If $Z + 1$ is the end of a new block of size w', and $T'_{B'(|B'|)}(1) \geq 0$, then append c_1 to the end of V, and append $\frac{Z+1}{w'}$ to the end of L and to the end of R. Also, call $DS_S.update(1)$.
5. If $Z + 1$ is the end of a new block of size w' and $T'_{B'(|B'|)}(1) < 0$, then, set $R(|R|)$ to $\frac{Z+1}{w'}$ and call $DS_S.update(0)$.
6. If $Z + 1$ is the end of a new block of size w', append c_1 to the end of P and append 0 to the end of B'.

It is easy to see that each step of the update algorithm, will take $O(1)$ time.

We will summarize the results from **lemma 7** and **lemma 8** into the following theorem:

Theorem 5. *There exists a data structure, that can store a binary array B of size Z, by using $O(Z)$ bits, and will support **rank**, **select**$_1$ and **access** operations in $O(1)$ time, and appending a bit to the end of the array in amortyzed $O(1)$ time.*

Proof. For array B, we will create an instance DS_1 of the data structure described in **lemma 7**, and an instance DS_2 of the data structure described in **lemma 8**. In order to answer $\mathbf{rank}_b$ and **access** queries, we will call $DS_1.\mathbf{rank}_b(i)$ or $DS_1.\mathbf{access}(i)$. For $\mathbf{select}_1$ queries, we will use the data structure DS_2. In order to update the current data structure, we will call $DS_1.update(v)$ and $DS_2.update(v)$.

As for every query and for every update, we will call a constant number of functions, each requiring at most $O(1)$ time, each query will require $O(1)$, and each update will require amortyzed $O(1)$ time.□

References

1. Chan, T.M., Durocher, S., Larsen, K.G., Morrison, J., Wilkinson, B.T.: Linear-space data structures for range mode query in arrays. Theory of Computing Systems **55**(4), 719–741 (2014)
2. Gu, Y., Polak, A., Williams, V.V., Xu, Y.: Faster monotone min-plus product, range mode, and single source replacement paths. arXiv preprint arXiv:2105.02806 (2021)
3. He, M., Liu, Z.: Exact and Approximate Range Mode Query Data Structures in Practice. In: Georgiadis, L. (ed.) 21st International Symposium on Experimental Algorithms (SEA 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 265, pp. 19:1–19:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2023). https://doi.org/10.4230/LIPIcs.SEA.2023.19, https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SEA.2023.19