Data Structures & Algorithms for Problem Solving (CS1.304)

Searching in Integer Data

Avinash Sharma

Center for Visual Information Technology (CVIT),

IIIT Hyderabad

Motivation - Integer Data

- Consider settings where it is known that the input keys arrive from a known range [0,m-1].
  - Example: Vertex ids in a graph algorithm
  - Marks in an exam out of 100.
  - Age, ...
- Suppose we are still interested in the set of operations such as insert/delete/find and in addition prev/next/isEmpty and max/min.
- This set of operations is called as the dictionary set of operations over dynamic set.
- Notice that a height-balanced BST can do each of these operations in O(log n) time.

Motivation - Integer Data

- However, the BST does not intend to benefit from the fact that the input keys are from a small known range.
- One possibility is to use an array of size O(m).
- Each operation can be done in O(m) time, some can be done in O(1) time.
- However, m can be closer to n, and hence, much bigger than O(log n).
- So, ideal solutions should do all operations in time less than O(log n).

Motivation

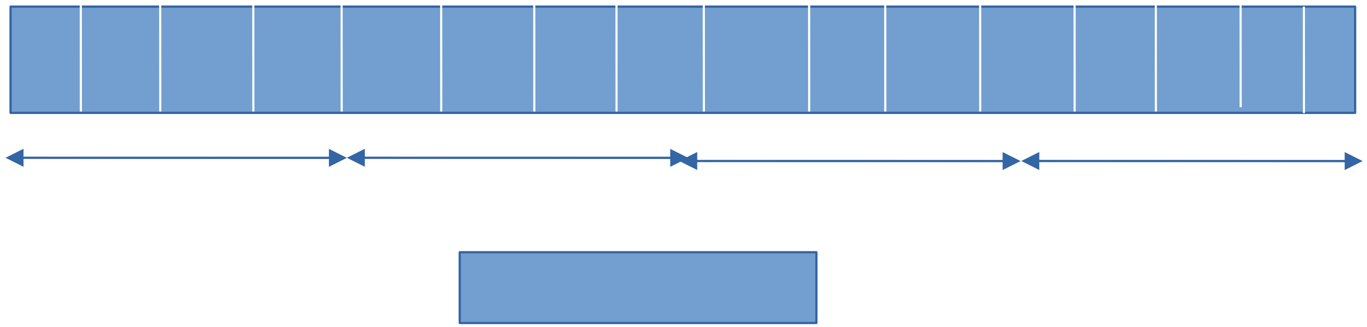
- We will see one data structure today that can do each of these operations in time O(loglog m).
  - The space used will be a bit high at O(m).
  - There are other ways to deal with that issue.
- The data structure is called the van Emde Boas tree after dutch computer scientist Peter van Emde Boas who invented it in 1974.
- We will take a two step approach.
  - First, to get to O(loglog m) for some operations and O(log m) for some,
  - Then, find ways to optimize the above solution and get to O(loglog m) for every operation.

Bit Vectors

0	0	1	1	0	1	0	0	0	1	0

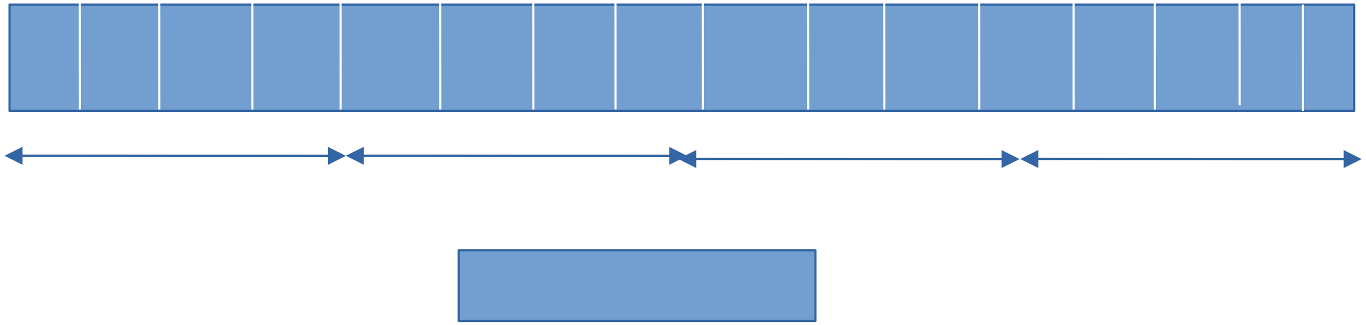
- By a bit vector, we mean an array where the values stored are 1 or 0.
- Usually, 1 in cell i means the presence of key i, and 0 means the absence of key i.

Tiered Bit Vectors



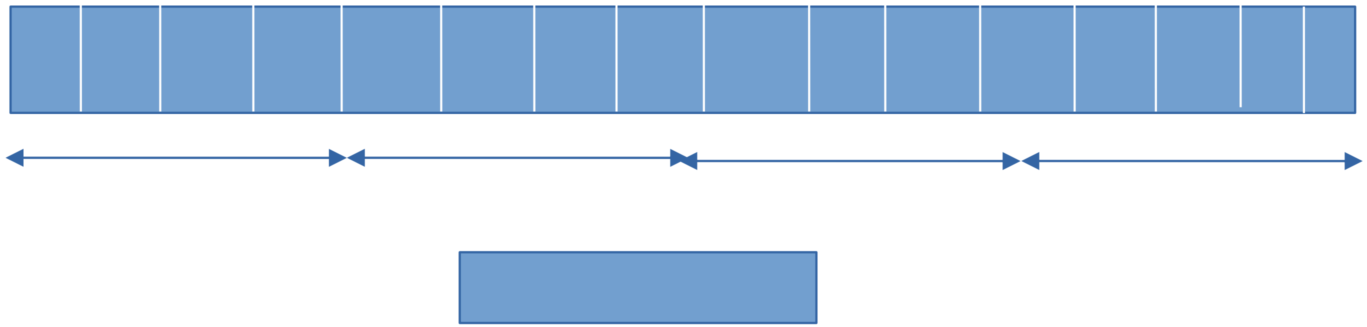
- While using one bit vector for the entire data structure is not efficient, we can keep a hierarchy of bit vectors.
- Call the first vector as A and the second bit vector as T.
- If T[i] is 1, it means that at least one element in the corresponding part of A is present.

Tiered Bit Vectors



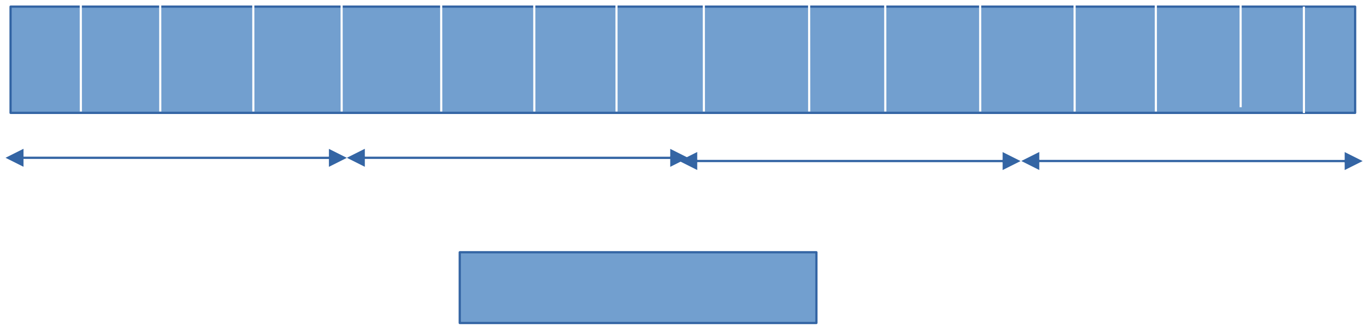
- Formally, let s be a parameter.
- The array A is partitioned into A/s pieces and the array T is of size s.

Operations on Tiered Bit Vectors



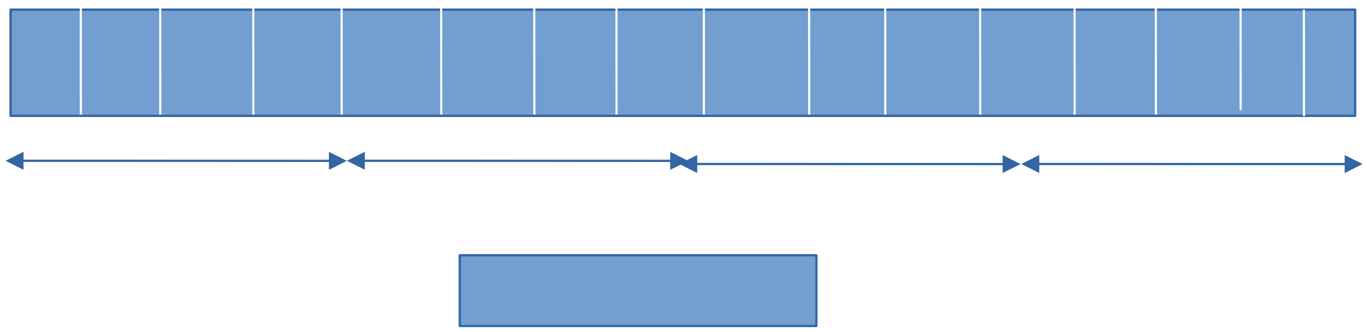
- Insert(x) – Two steps
  - Check if T[x / s] is 0. If it is, then make T[x / s] as 1.
  - Mark A[x] as 1. Can be viewed alternatively as inserting in the array $A_{i/s}$.
- Each Insert(x) is therefore translated to two Insert calls into appropriate vectors (arrays).

Operations on Tiered Bit Vectors



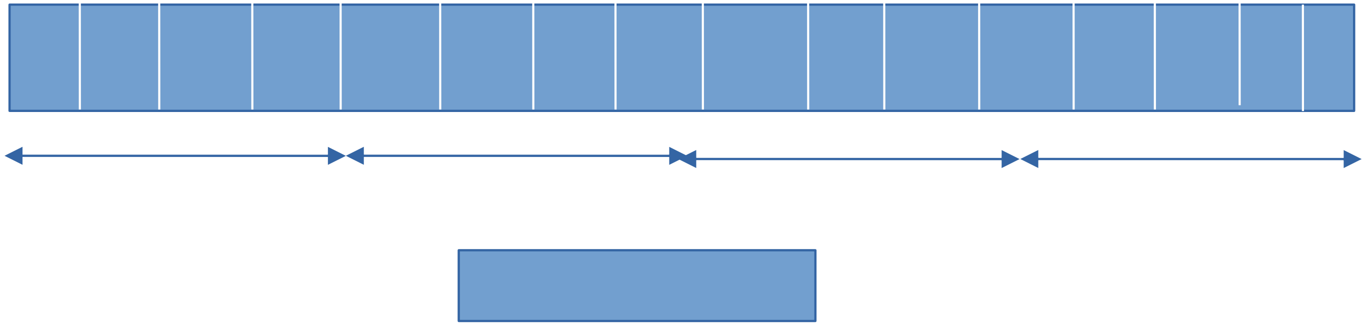
- Find(x) – Two steps
  - Otherwise, check for x in array $A_{x/s}$.
- Each Find(x) is therefore translated to one isempty() query and one find on a small array.

Operations on Tiered Bit Vectors



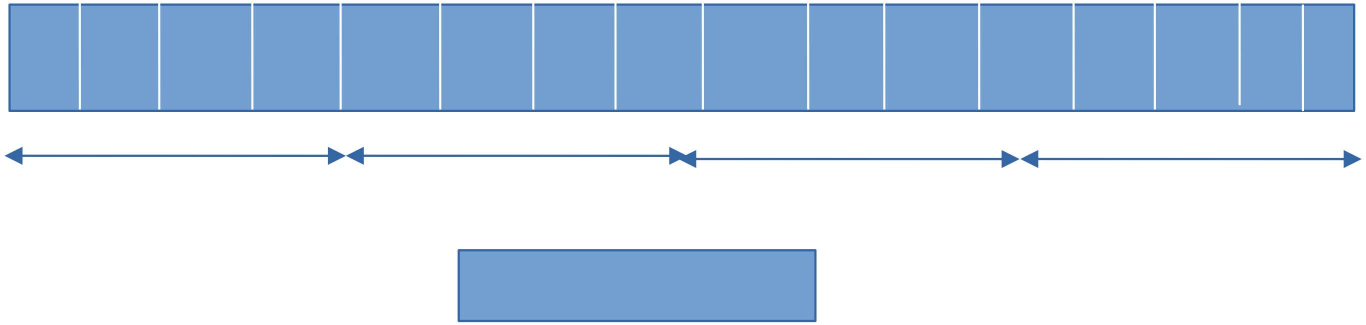
- Min() should return the value of the smallest present element.
- Find the minimum value in the smaller vector, say j.
  - Suggests that T[j] is 1 and all other T[i] if i < j are 0.
- Find minimum in A_j.
- In essence, Min() translates to two Min() operations on smaller vectors.

Operations on Tiered Bit Vectors



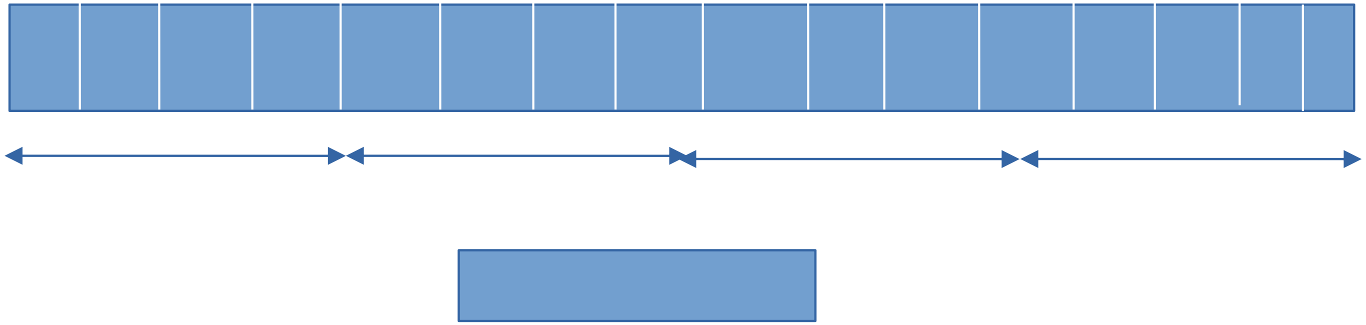
- Prev(x) – Again multiple steps on smaller vectors.
  - Find Min() in the vector $A_{x/s}$.
  - If the answer is different from x, return.
  - Otherwise, find j = Prev(x/s) in the smaller vector.
  - Find Max() in A_j.
- Each Prev(x) is translated to two Prev() queries. Same with Next(x).

Operations on Tiered Bit Vectors



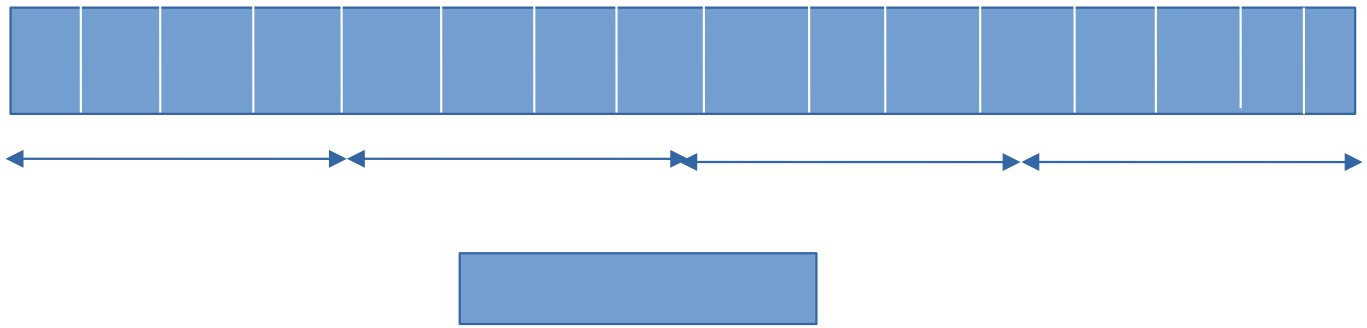
- Delete(x) – Again two steps
  - Mark A[x] in $A_{x/s}$ as 0.
  - If all entries in $A_{x/s}$ are 0, then set T[x/s] to 0.

Operations on Tiered Bit Vectors



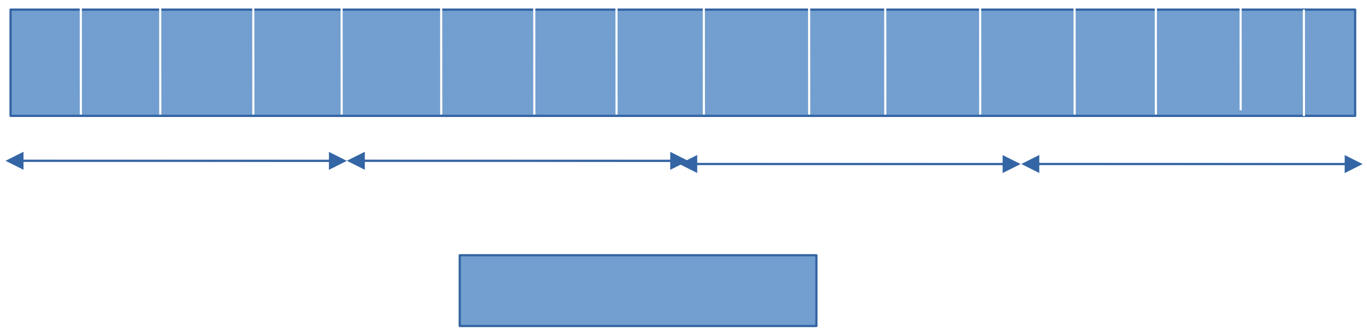
- Each operation turns into operations on smaller arrays as follows.
  - insert: 2x insert
  - find : 1x lookup
  - is-empty: 1x is-empty
  - min: 2x min
  - prev: 2x Prev, 1x max, and 1xmin
  - delete: 2x delete, 1x is-empty

Operations on Tiered Bit Vectors



- Each operation turns into operations on smaller arrays as follows. Time taken is:
  - insert: 2x insert – O(1)
  - find : 1x lookup – O(1)
  - is-empty: 1x is-empty – O(1)
  - min: 2x min – O(s + m/s)
  - next: 1x next, 1x max, 1x min – O(s+m/s)
  - delete: 2x delete, 1x is-empty – O(s+m/s)

Operations on Tiered Bit Vectors



- Each operation turns into operations on smaller arrays as follows. Time taken is:
  - The run time of O(s+m/s) is seen to be minimized when s = sqrt{m}.
  - In other words, all operations finish in time O(sqrt{m}).
    - Some operations are much faster.
    - Not a good solution – BST much better.

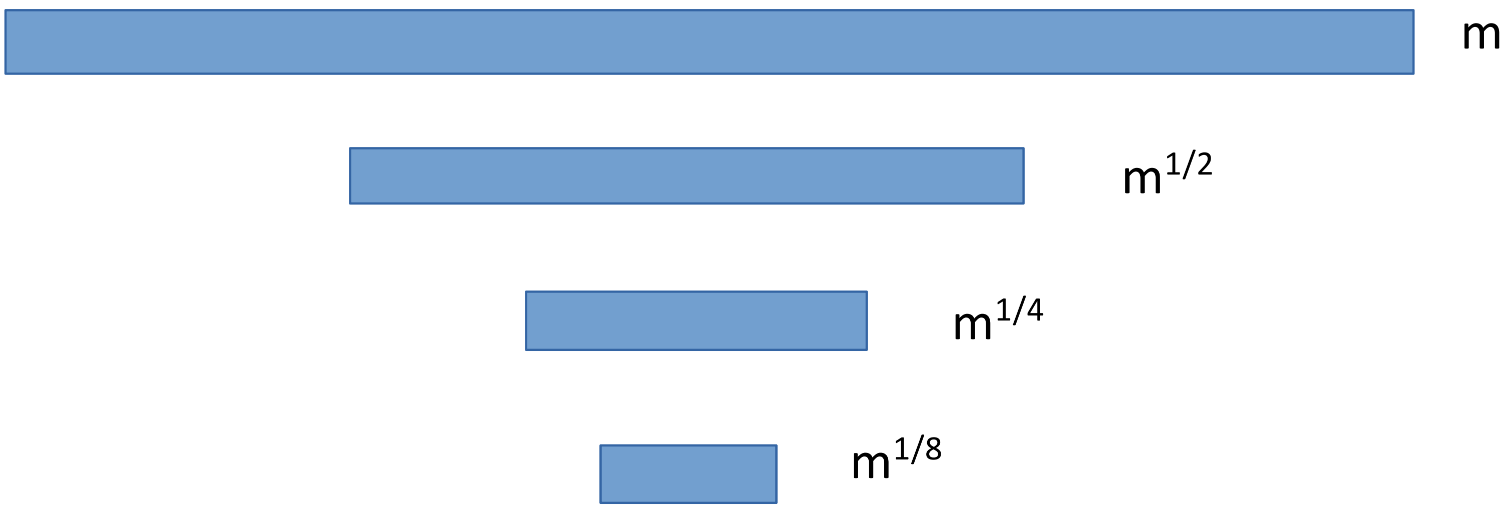
Extending Tiered Bit Vectors

- Consider the second bit vector T of size sqrt{m}.
- Since sqrt{m} is much bigger than log m in general, our run times are all large.
- But, what prevents us from using additional techniques for storing a summary of this bit vector?
  - Nothing. Can apply our technique recursively.

Extending Tiered Bit Vectors

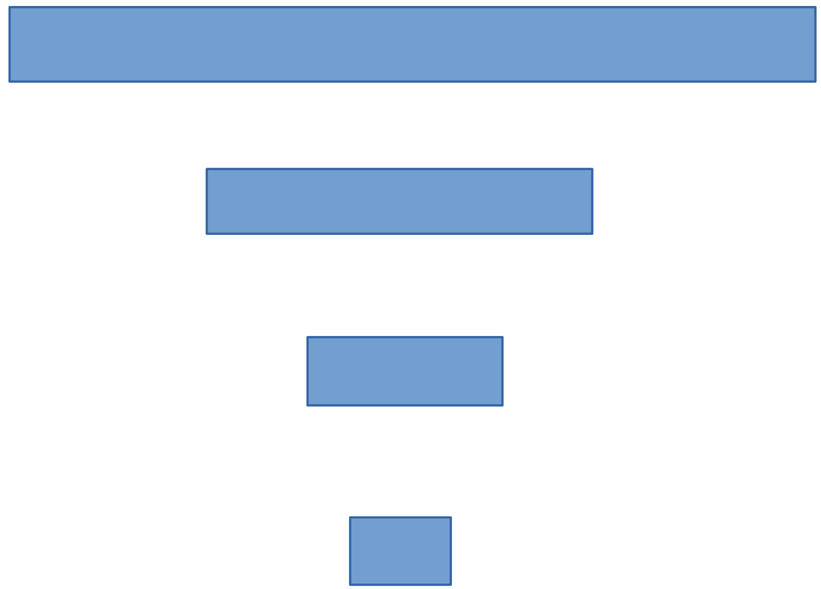
- Let us call the large vector of size m, A, as T_0 and summary vector as T_1 of size $m^{1/2}$.
- Consider using a summary vector for vector T_1.
- The length of this summary vector should be $m^{1/4}$. Call this vector as T_2.
- Since T_2 is also large, possibly, we will use a summary vector T_3 to help operations on T_2.
  - The size of T_3 is $m^{1/8}$.
- Continue this until the summary vector is of size O(1).
  - We will see that there will be O(log log m) vectors.

Extending Tiered Bit Vectors



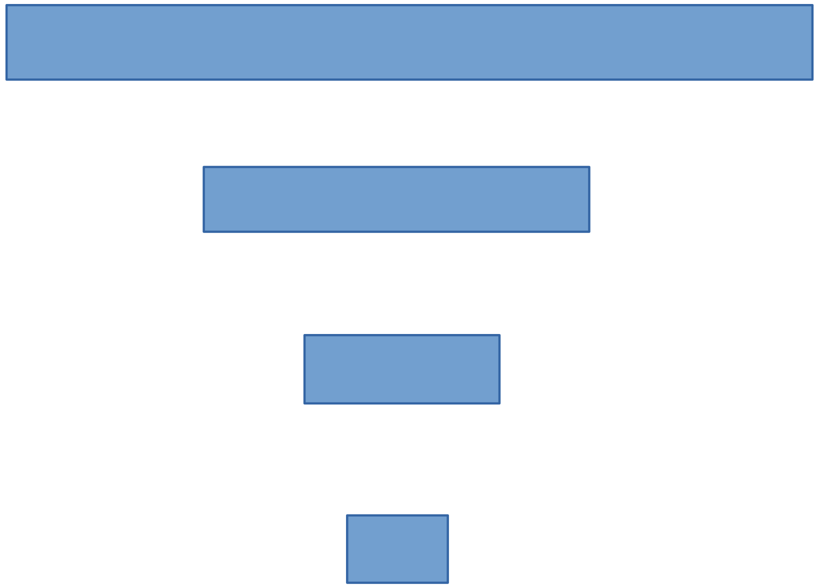
Extending Tiered Bit Vectors

- A helpful view is to see the two levels of the hierarchy as forming $1+m^{1/2^i}$ smaller vectors at level i.



Operations on Extended Tiered Bit Vectors

- Consider the set of operations again.
  - Insert: 2x insert, at each recursive level.
  - Find : 1x lookup – but can say lookup in a smaller bit vector.
  - is-empty: 1x is-empty, but can say on a smaller bit vector
  - min: 2x min, one each at a smaller level
  - next: 1x next, 1x max, 1x min, at a smaller level
  - delete: 2x delete at a smaller level, 1x is-empty,

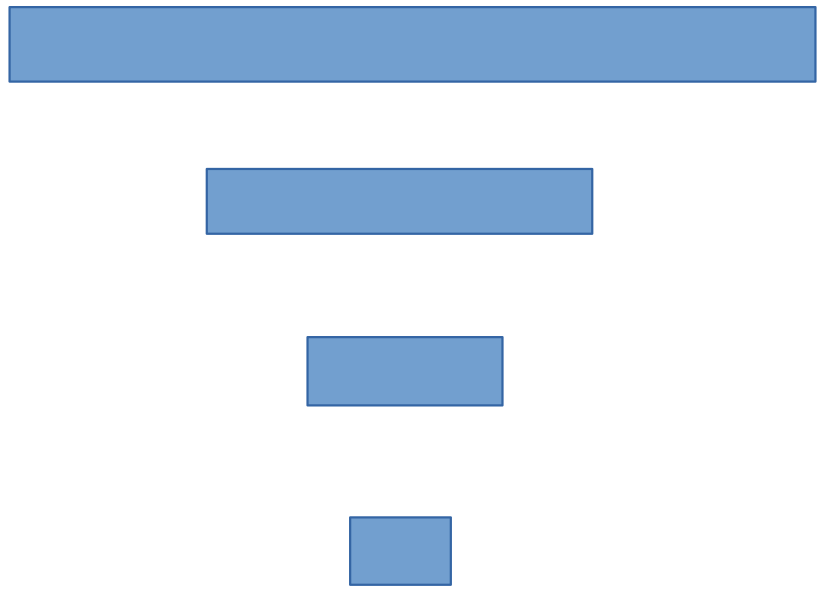


Operations on Extended Tiered Bit Vectors

- Consider the set of operations again.
  - Insert: 2x insert, at each recursive level.

    Time = 2T(sqrt{m}) + O(1)

    Solution: T(n) = log(m)



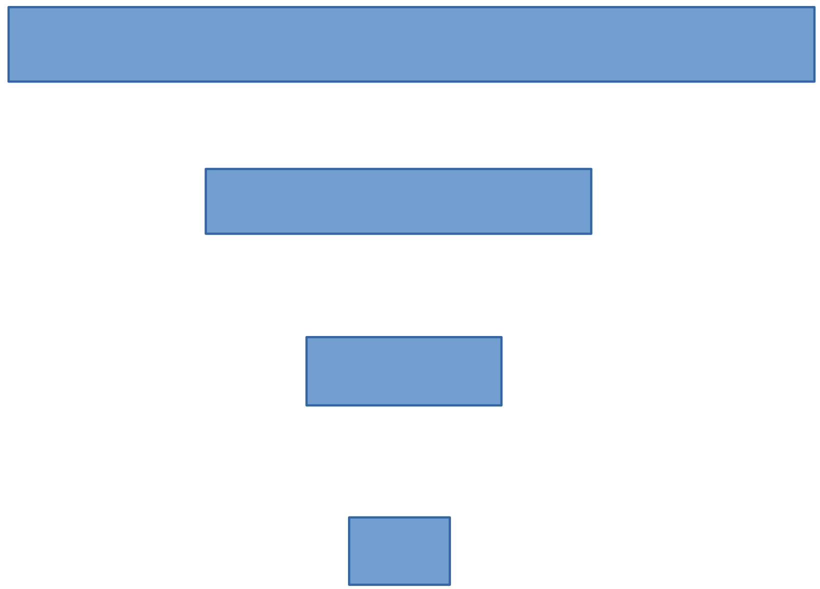
Extending Tiered Bit Vectors

Proposition 1. $T(U) = T(\sqrt{U}) + O(1) = O(\log\log U)$.

Proof. Let $m = \log U \Rightarrow U = 2^m$. Our recurrence relation is now: $T(2^m) = T(2^{\frac{m}{2}}) + O(1)$. Let $S(m) = T(2^m)$, then we have that: $S(m) = S(\frac{m}{2}) + O(1)$. By case 2 of the master method, $S(m) = O(\log m)$. Therefore, $T(U) = T(2^m) = S(m) = O(\log m) = O(\log\log U)$. [**?**] $\square$

Operations on Extended Tiered Bit Vectors

- Consider the set of operations again.
  - Find : 1x lookup – but can say lookup in a smaller bit vector.
  - is-empty: 1x is-empty, but can say on a smaller bit vector
  - min: 2x min, one each at a smaller level
  - successor: 1x successor, 1x max, 1x min, at a smaller level
  - delete: 2x delete at a smaller level, 1x is-empty,

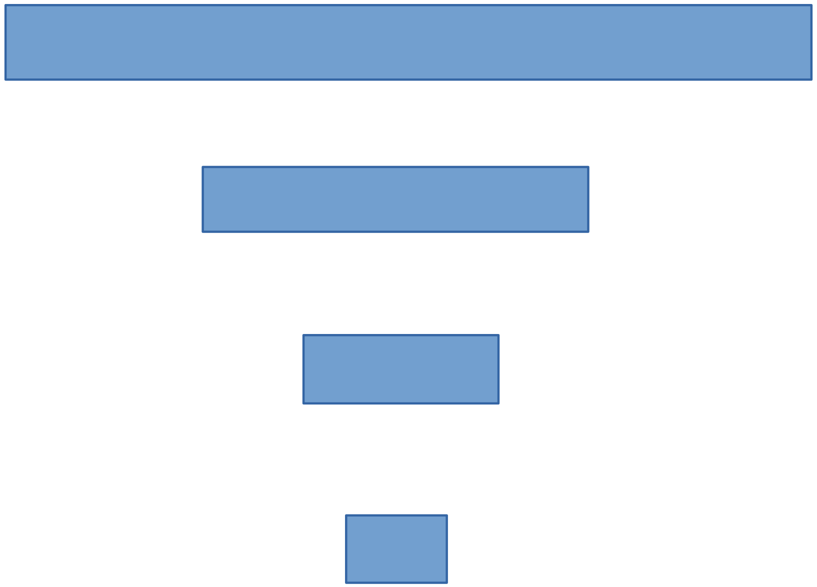


Operations on Extended Tiered Bit Vectors

- Consider the set of operations again.
  - Find : 1x lookup – but can say lookup in a smaller bit vector.

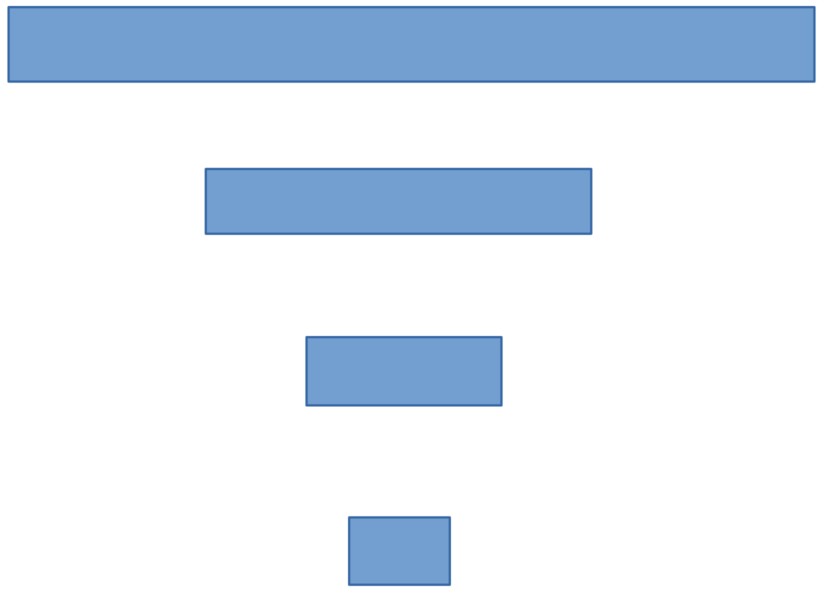
    Time : T(m) = T(sqrt{m}) + O(1)

    Solution: T(m) = O(loglog m).



Operations on Extended Tiered Bit Vectors

- Consider the set of operations again.
  - is-empty: 1x is-empty, but can say on a smaller bit vector
    Time: $T(m) = T(\sqrt{m}) + O(1)$.
  - min: 2x min, one each at a smaller level
    Time: $T(m) = 2T(\sqrt{m}) + O(1)$
    Solution: From earlier, $T(m) = \log(m)$



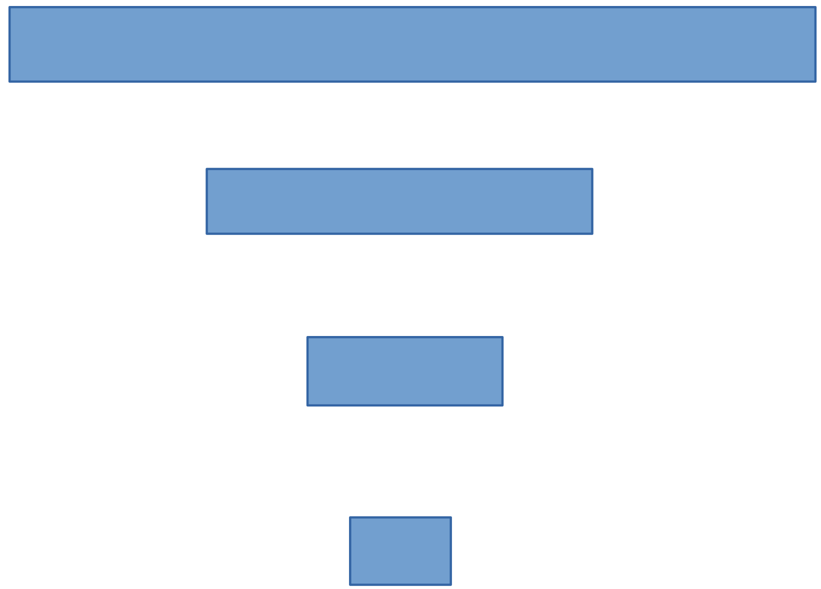
Operations on Extended Tiered Bit Vectors

- Consider the set of operations again.
  - next: 1x next, 1x max, 1x min, at a smaller level
  - delete: 2x delete at a smaller level, 1x is-empty,

T(m) = T(sqrt{m}) + T(max) + T(min)

= T(sqrt{m}) + O(log m)

= O(log m)



Our Solution So Far

- We have a data structure, the tiered bit vector, where some operations are O(loglog m).
  - These are exponentially faster than using BSTs.
- Other operations are in O(log m).
  - These are worse than BST.
- Can we make all operations faster?
  - Let us identify some potential places for optimization.

Scope for Optimization

- Notice that our recurrences that lead to slower run times are of the form $T(m) = 2T(\sqrt{m}) + O(1)$.
- We will see if we can use do with just one $T(\sqrt{m})$ on the RHS instead of two of them.
- For this, let us start with Min/Max/isEmpty.

Faster Min/Max

- Finding the smallest or the largest should be O(1).
- Even when there is a data structure that does lots of other operations, min/max can be found in O(1).
- The trick is to store the minimum and maximum explicitly outside of the data structure.
  - Operations that affect min/max can check before proceeding to the data structure.

The Van Emde Boas Tree

- Before we see the impact of this reduced cost of Min/Max, let us define the van Emde Boas Tree formally.
- The van Emde Boas tree of size m is a bit vector of size m that has
  - 1 + sqrt{m} vectors of size sqrt{m} with pointers to each of these vectors.
  - Two separate data items, for the min and the max of the current level.
  - Each vector, including the summary, is again stored recursively via vectors of size $m^{1/4}$.

The Van Emde Boas Tree

|U| = 16

Summary

1	0	0	1

0	1	0	0

Cluster[0]

0	0	0	0

Cluster[1]

0	0	0	0

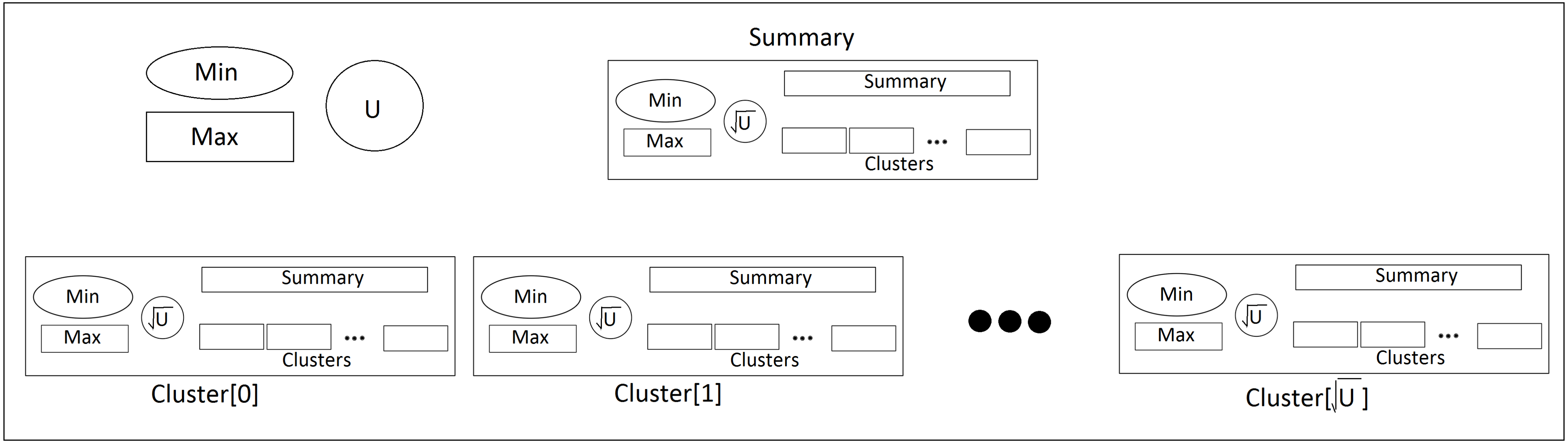
Cluster[2]

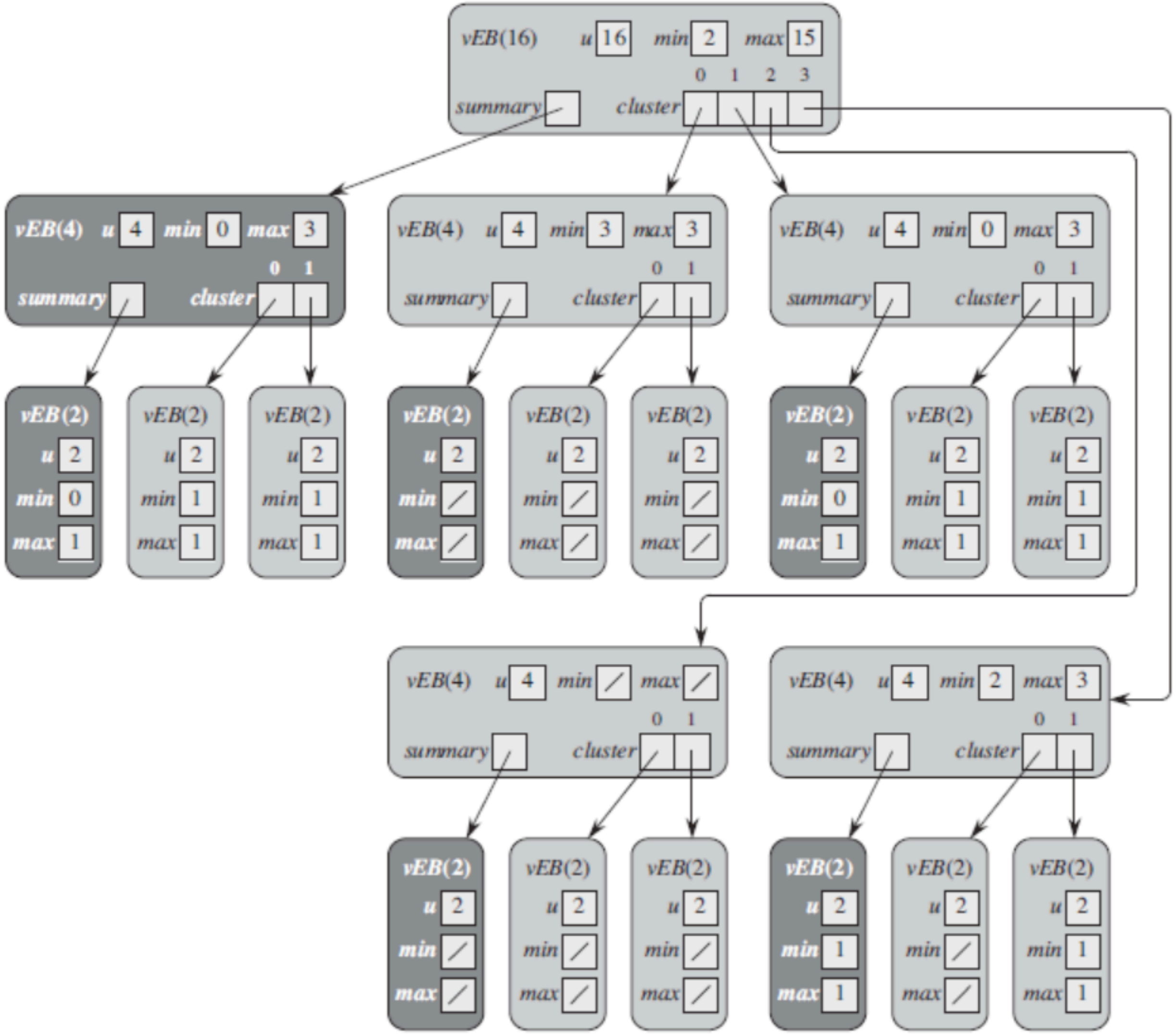
0	0	1	0

Cluster[3]

The Van Emde Boas Tree

The Van Emde Boas Tree




vEB(16) u 16 min 2 max 15
summary
cluster 0 1 2 3
vEB(4) u 4 min 0 max 3
vEB(4) u 4 min 3 max 3
vEB(4) u 4 min 0 max 3
vEB(4) u 4 min max
vEB(4) u 4 min 2 max 3
vEB(2)
u 2
min
max

The Van Emde Boas Tree

- Let us look at Insert operation in detail.
- Insertions in a vEB tree work as earlier.
  - Careful to handle min and max.
  - Handle the case where the tree is empty.
  - Handle the case where the tree has just one element.
  - May need to display min or max into the tree

The Van Emde Boas Tree

```
INSERT(x, V)
if V.min == +∞ then
    V.min = x
    return
else if x < V.min then
    SWAP(x, V.min)
end if
if x > V.max then
    V.max = x
end if
if V.cluster[c].min == +∞ then
    INSERT(i, V.cluster[c])
    INSERT(c, V.summary)
else
    INSERT(i, V.cluster[c])
end if
```

1. If V is empty, i.e., $V.min == +\infty$, then we only need to set $V.min = x$.
   $\Rightarrow O(1)$
2. Else if $x < V.min$, then x is the new minimum element in V. Hence, we swap x and $V.min$ and continue with the insertion with our new x value.
   $\Rightarrow O(1)$
3. If $x > V.max$, then x is the new maximum element in V. Unlike $V.min$, $V.max$ is also stored in the clusters, thus we simply set $V.max = x$ and proceed with the insertion.
   $\Rightarrow O(1)$
4. If $V.cluster[c]$ is empty, i.e., if $V.cluster[c].min == +\infty$, we need to insert i into $V.cluster[c]$ and also insert c into $V.summary$. Here we need to execute two recursive calls on a universe size of $\sqrt{U}$. However since $V.cluster[c]$ is empty, to insert i into $V.cluster[c]$ we only need to set $V.cluster[c].min = i$ which is $O(1)$ time.
   $\Rightarrow$ Two recursive calls on a problem size of $\sqrt{U}$, with one recursive call taking $O(1)$ time.
5. Else we just need to insert i into $V.cluster[c]$.
   $\Rightarrow$ One recursive call on a problem size of $\sqrt{U}$

The Van Emde Boas Tree

- Time to insert analyzed as follows.
  - If the tree is empty or has just one element, update min and max appropriately and stop.
  - Insert x % sqrt{m} into the appropriate vector ($A_{x/m}$).
  - Insert x / sqrt{m} into the summary vector.
- Still has 2 recursive inserts and O(1) time.
- Not much help.
- Notice however that the second insert, into the summary vector, is not needed on every insert.
  - Needed only when the corresponding summary vector is empty so far.

The Van Emde Boas Tree

- New recurrence relation:
  - $T(m) = T(\sqrt{m}) + O(1)$ when no insert into the summary. Otherwise,
  - An insert into an empty vector followed by an insert into another vector.

The Van Emde Boas Tree

- New recurrence relation:
  - T(m) = T(sqrt{m}) + O(1) when no insert into the summary. Otherwise,
  - An insert into an empty vector followed by an insert into another vector.
    - Inserting into an empty vector is O(1).
      - If the tree is empty or has just one element, update min appropriately and stop.
    - The recurrence in this case is T(m) = T(sqrt{m}) + O(1).
  - In both cases, the time to insert is O(log log m).

The Van Emde Boas Tree

- TODO: Analyze other operations.
- You can then conclude that each operation runs in time O(loglog m).

Thank You