Explanation.

permutation → $[1 \; 2 \; 3 \; 4 \; 5 \; \ldots \; m]$

queries ⇒ $[k_1, k_2, k_3, k_4, \ldots]$.

~~Given~~ When queries[i] comes to us, we ~~can~~ ~~search~~ have to tell the position, where it is in the permutation order. And then we have to move that in front of the permutation.

query(k)

$[1 \; 2 \; 3 \; 4 \; \ldots \; k \; \ldots \; m]$

$[k \; 1 \; 2 \; 3 \; 4 \; 5 \; \ldots \; m]$

Given the constraints $m \le 10^3$ and

queries $\le m$.

Even if ~~on~~ we ~~make~~ simulate the whole process & for a given $\underline{k}$ just ~~ma~~ store the permutation in a LinkedList

1 → 2 → 3 → 4 → . . . → k → - - - → m

Then ~~we~~ for search just travel the Linked List, Move ~~the~~ it to the front of the Linked List. Basic linkedlist operation.

For each query $O(m)$. total number of queries m.

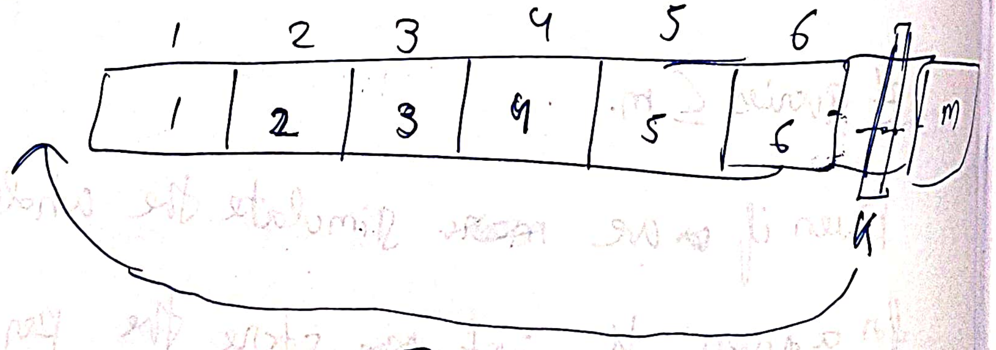
$$m \times m \Rightarrow O(m^2).$$

(Better Approach)

Use BIT tree, using bit tree we will be able to ~~tell~~ ~~where~~ ~~is the~~ ~~inde~~ use faster queries.

Intuition

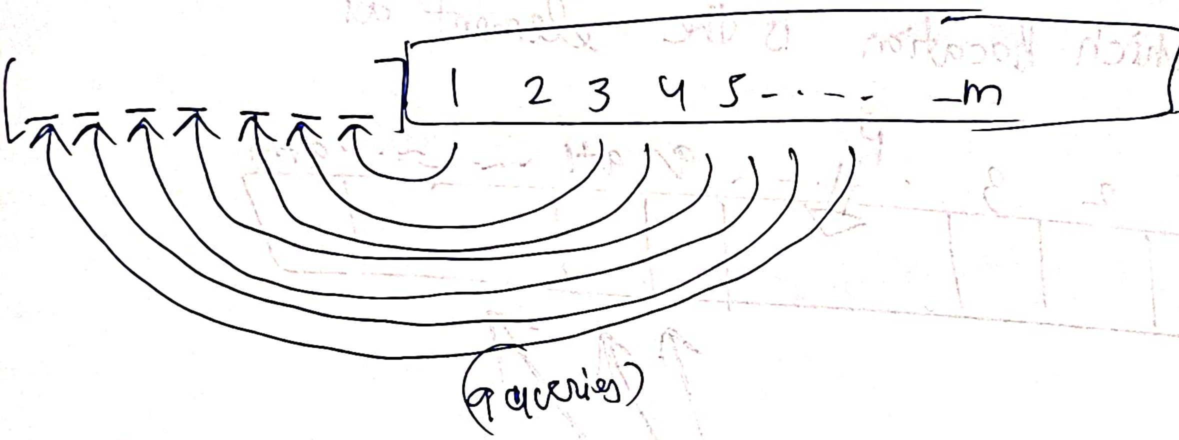
initial permutation



when any query is performed. lets say k.

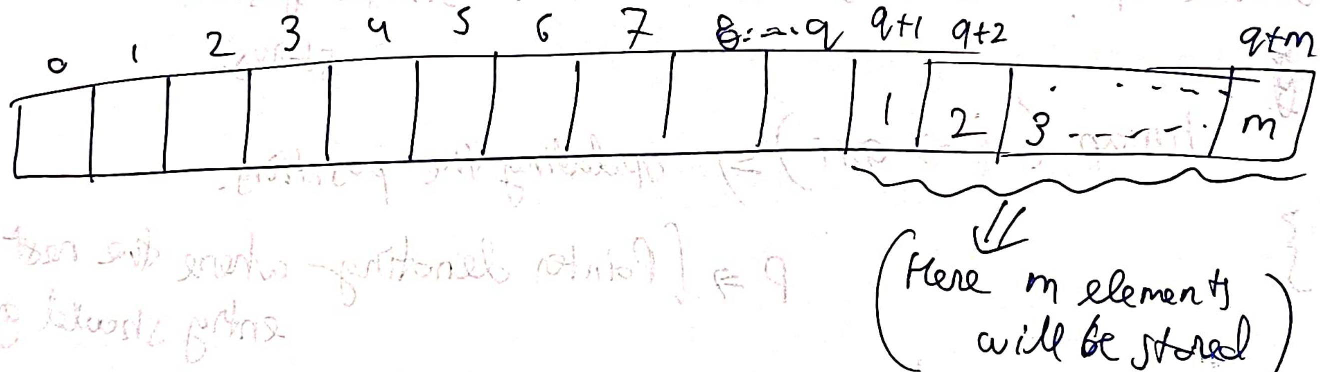
Lets say total number of queries = q.

So we know q times any number will come to the front.

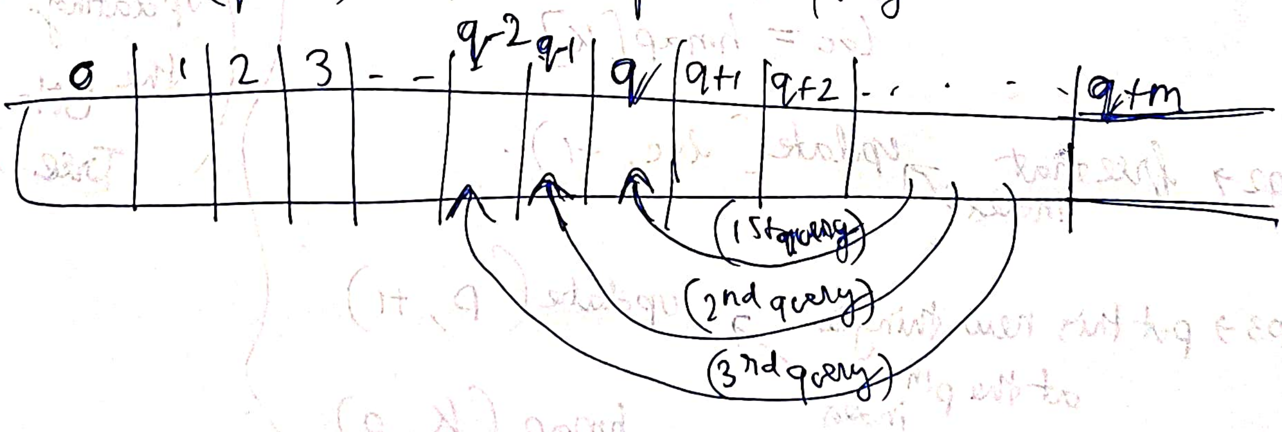


So we create a BIT tree of size [q+m+1]

+1 → for 1 based indexing

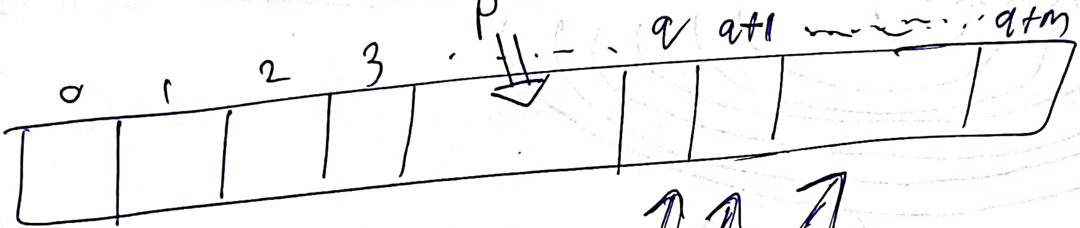


For each query operation we can move the element to $(q+1-i)^{th}$ position for i^{th} query



Instead of storing the actual numbers in the array of BIT tree we will store 1 and 0.

And we will have a HashMap < Number → Location) to denote which location is the element at.



for (int i = 1; i <= m; i++){

Bittree . update (q+i, 1);

hmap (i → q+i) ⇒ updating the positions.

}

making updates and storing in the range fashion that it stores.

p ⇒ [Pointer denoting where the next entry should go]

➤ Part when a query comes → (K)

Step1 → Get the location from map.

loc = hmap[K]

Step2 → free that index → update(loc, -1).

Step3 → put this new thing at the pth index → update (p, +1)

hmap (K, p)

p --;

(Updating the Bit Tree)

Interesting part comes, on how to new tell its location.

My thoughts →

Am thinking. Since in HashMap we have

its location. loc = hmap.get(k).

Now this loc is of the ~~BIT tree~~ new Array

that we formed.

we have to find the relative index. The one which

it has in the modified permutation.

option 1 → sum (loc) → tells me exactly how many

+1's are there from (1 to Loc) that should be the

position of K.

option 2 →