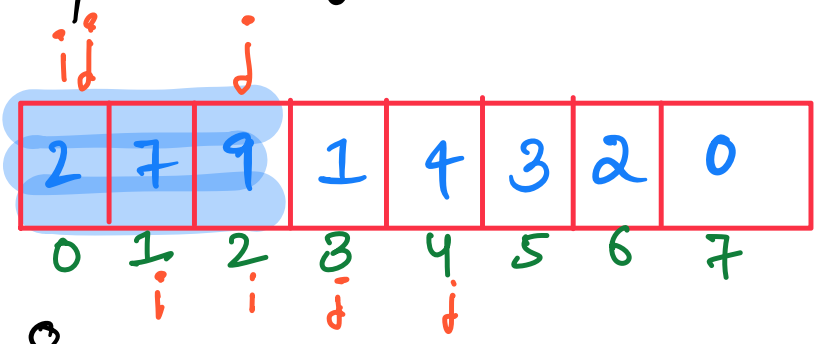
Sliding Window

Sliding Windows ★★★★

1) fixed size sliding window ✓
2) Dynamic Size Sliding window

Example

Give an array of length "N" find the Sum of every Subarray of length "K"



Length = 8

K = 3

Idea 1

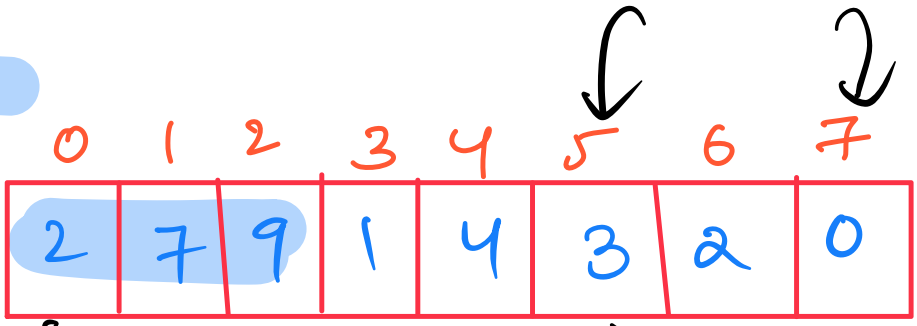
```
ds=[ ]
for(int i=0 ; i<=n-k ; i++){
    int sum=0;
    for(int j= i ; j<i+k ; j++){
        sum+= A[j];
    }
    ds.pb(sum);
}
```

Time complexity
O(n*k)

$1 \le n \le 10^5$
$1 \le k \le 10^5$] → $O(n*k) \rightarrow O(10^5 \times 10^5)$

$= O(10^{10})$ TLE

Idea 2



$k = 3$

ds = [18, 17, 14, 8]

Sum = ~~0 2 9 10 16 17 10 14 15 8 7 9~~ 5 9

Size = $j - i + 1 = 1 - 0 + 1$

$= 2$

$= 2 - 0 + 1$

$= 3$

$O(n)$ ✓

```
int i=0, j=0;
// find first k sized window Sum
Loop (i -> 0 to k)
    Sum += A[i]
```

Useful for array/string based problems with a constant Subarray size

When to use?

⇒ Calculating some Information for every fixed length Subarray in an array

⇒ Use of two Pointers

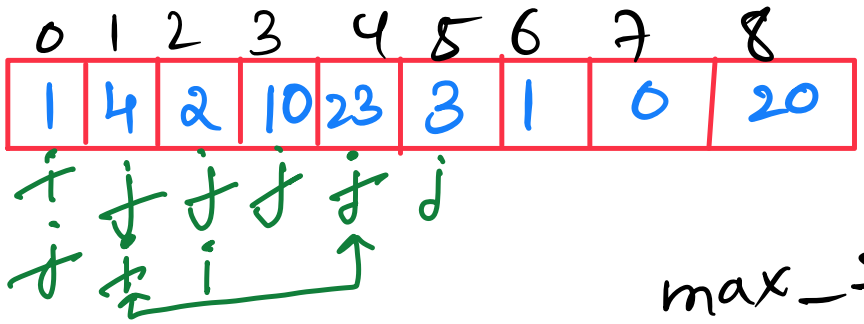
⇒ Very useful for interviews too

problem 1

find the Subarray of size k which has maximum Sum in an array

I/p $A = \{1, 4, 2, 10, 23, 3, 1, 0, 20\}$

$k = 4$



max_sum = ~~17~~ ~~0~~ 39

Sum = ~~0~~ 1 ~~5~~ ~~7~~
~~17~~ ~~16~~ 39
~~25~~ 38

Time :- O(n)
Space: O(1)

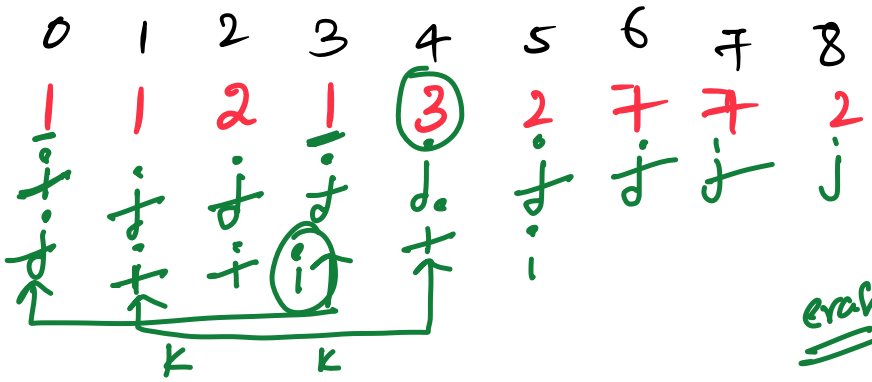
Problem 2

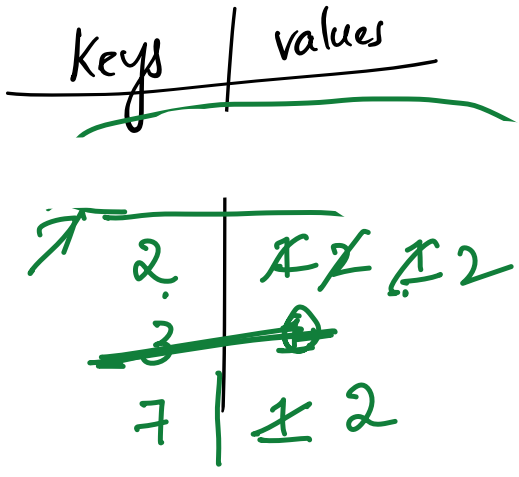
Find the Subarray of Size "K" which has maximum distinct elements in an array

I/P : arr = {1, 1, 2, 1, 3, 2, 1, 1, 2}

k = 4

O/P : 3 ⇒ {1, 3, 2, 1} hash map





evak

max_Dist = ~~0~~ ~~2~~ ~~3~~ 4

Time Complexity :-

$O(n) * O(\log_2(k))$

↑ Why "k"

Why not "n"

Problem 3

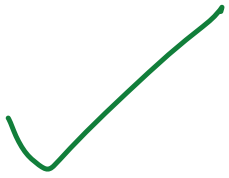
find the Subarray of Size k which has maximum Sum and all distinct elements

I/p $A = \{1, 2, 1, 3, 2, 1, 1, 2\}$ $k = 3$

o/p : 6

① All distinct

② max Sum



Window max

nums = { 1, -1, -1, -3, -5, 3, 6, 7 } k = 3

(-3,3) (-1,2)

① ②

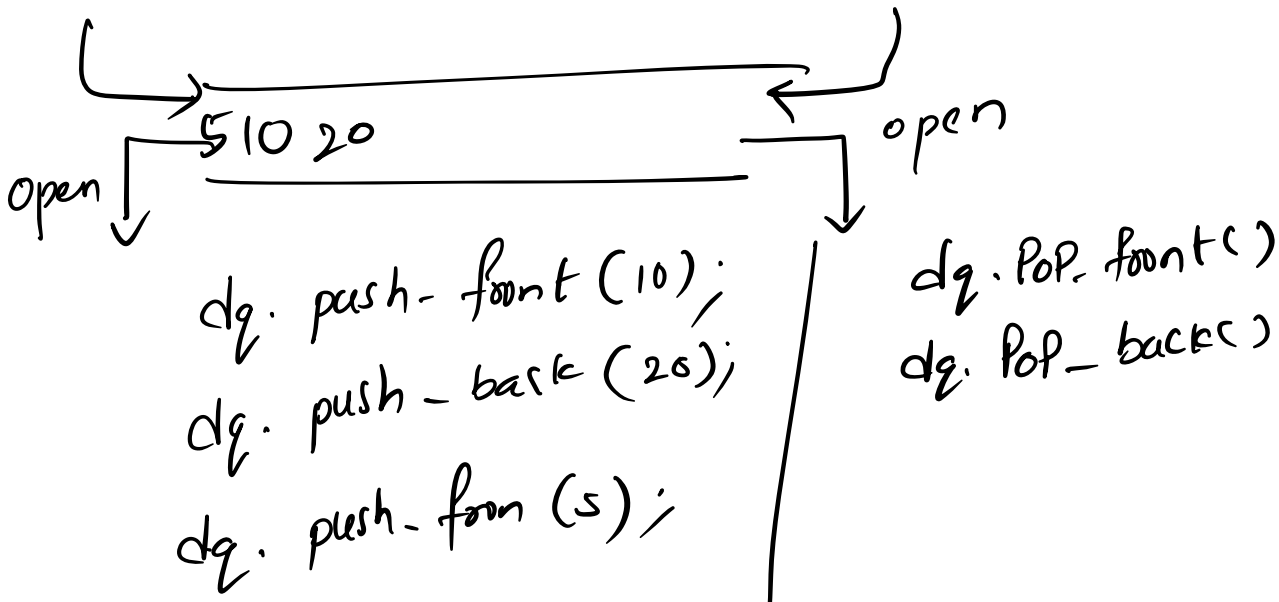
Time :-

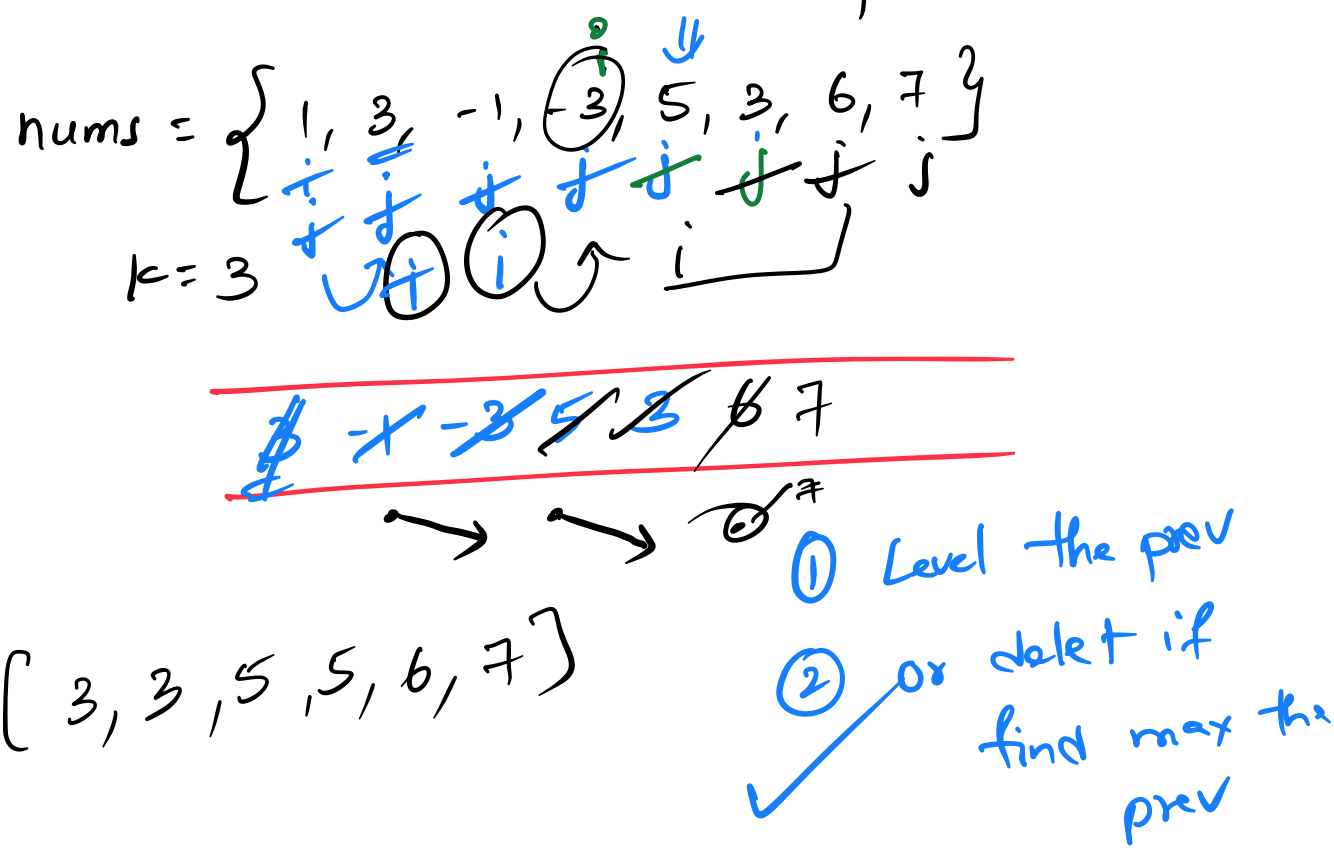
O(n) * O(log2(k))

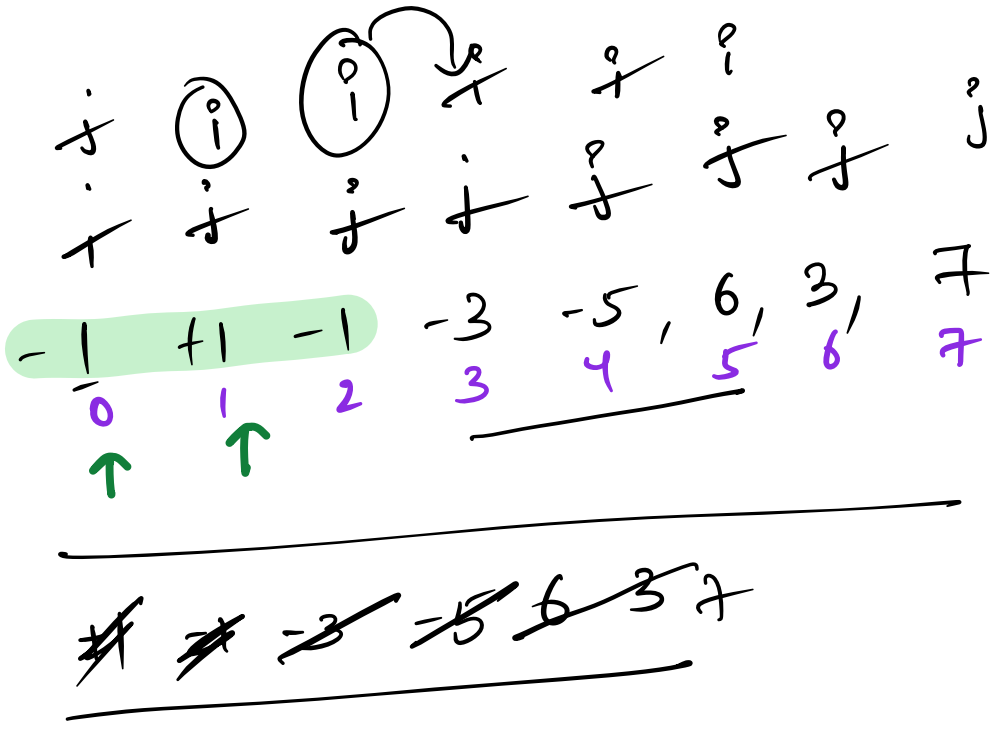
* Best Solution

Queue / deque.

the elements which are added first in the window will be removed first , so some times we can optimize our codes by using queue / deque insted of sets

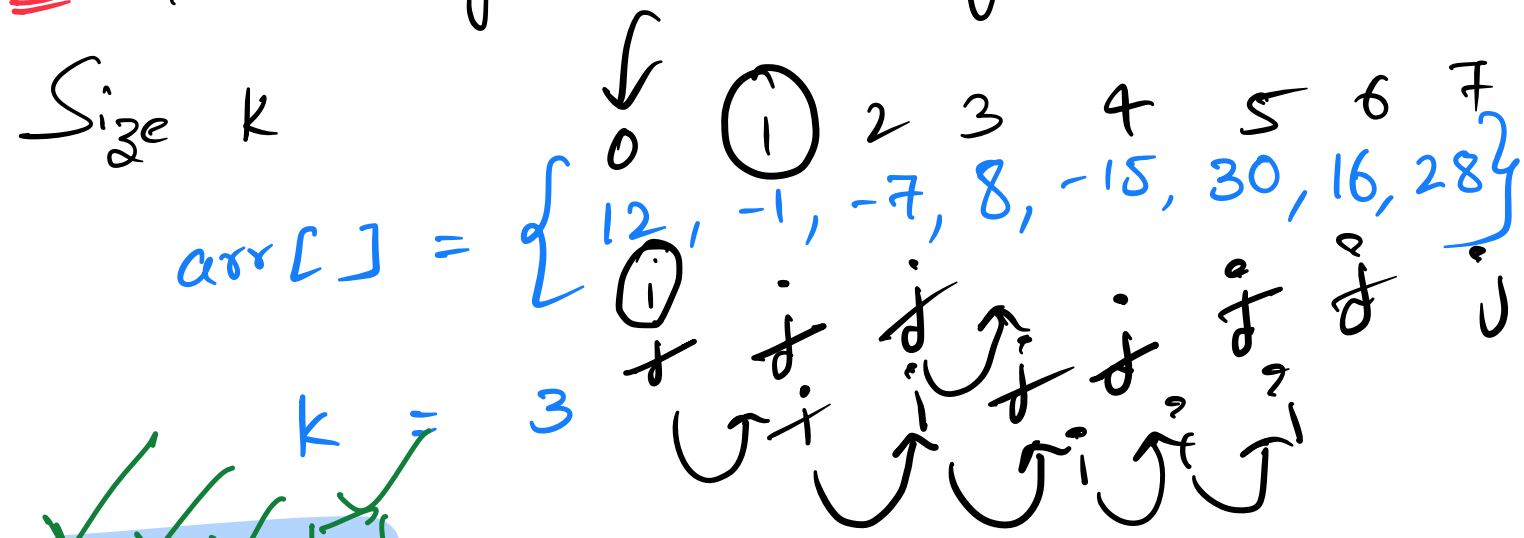


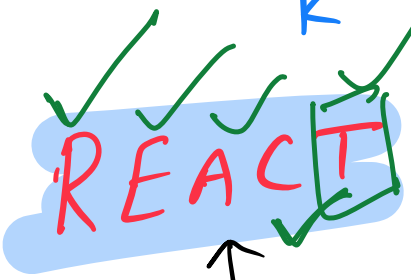

nums = { 1, 3, -1, -3, 5, 3, 6, 7 }
k = 3
[3, 3, 5, 5, 6, 7]
① Level the prev
② or delet if find max the prev


-1 +1 -1 -3 -5 6 3 7
0 1 2 3 4 5 6 7

① ① -1 6 6 7

Q first negative in Every window of Size k





sliding window + Queue

~~0 1 4~~

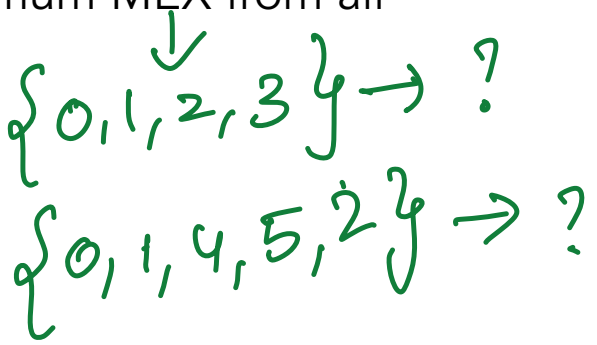
ans = [-1, -1, -7, -15, -15, 0]

Time : O(n)

Space : O(K) ← element in Q

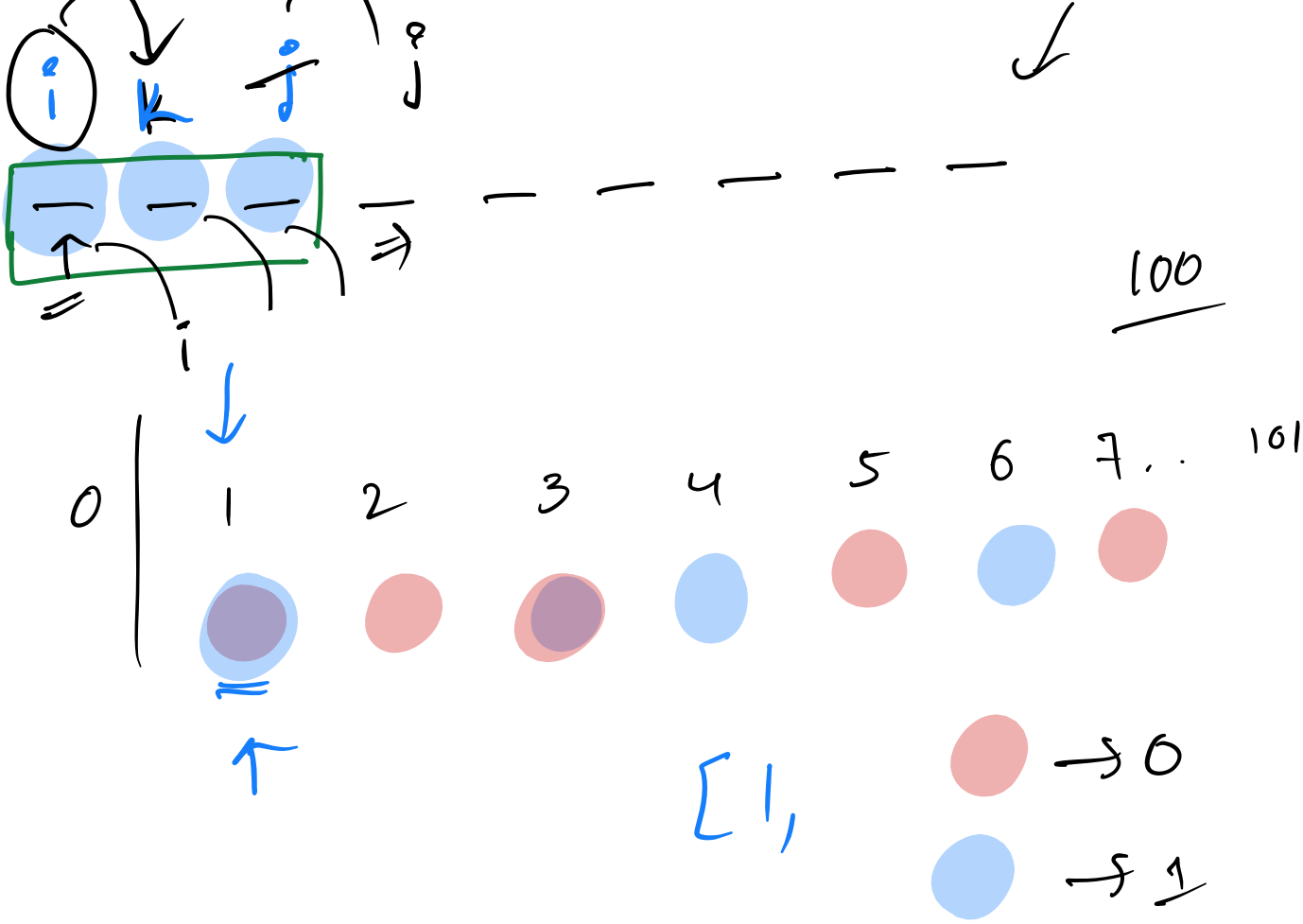
Maximum MEX from all subarrays of length K

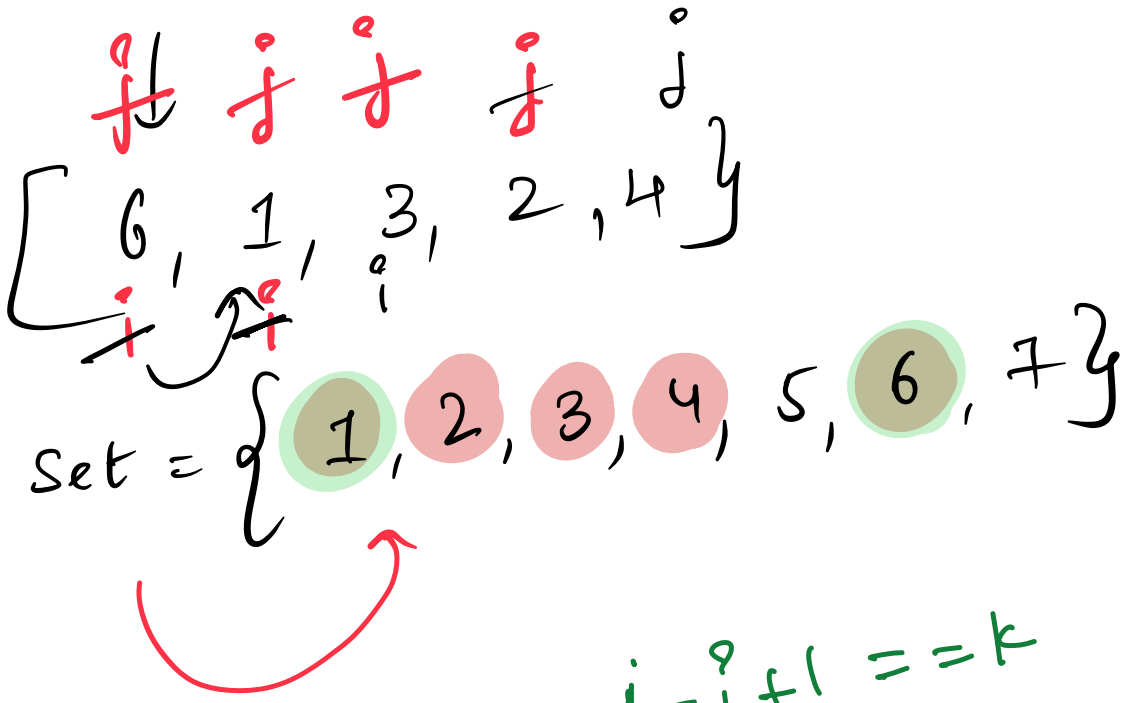
Given an array arr[] consisting of N distinct integers and an integer K, the task is to find the maximum MEX from all subarrays of length K.



Input: arr[] = {3, 2, 1, 4}, K = 2
Output: 3
Explanation:
All subarrays having length 2 are {3, 2}, {2, 1}, {1, 4}.
In subarray {3, 2}, the smallest positive integer which is not present is 1.
In subarray {2, 1}, the smallest positive integer which is not present is 3.
In subarray {1, 4}, the smallest positive integer which is not present is 2.

Input: arr[] = {6, 1, 3, 2, 4}, K = 3
Output: 4
Explanation:
All subarrays having length 3 are {6, 1, 3}, {1, 3, 2}, {3, 2, 4}
In subarray {6, 1, 3}, the smallest positive integer which is not present is 2.
In subarray {1, 3, 2}, the smallest positive integer which is not present is 4.
In subarray {3, 2, 4}, the smallest positive integer which is not present is 1.

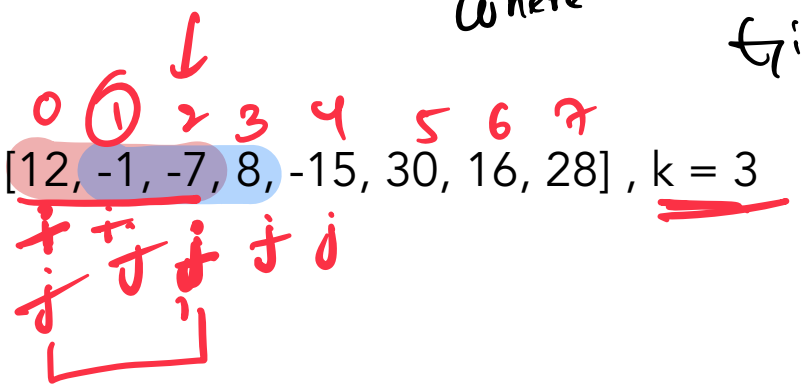




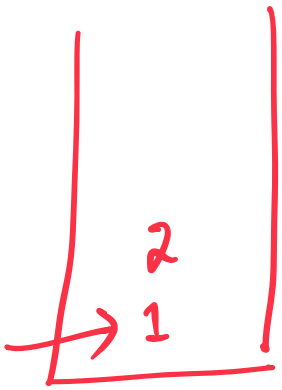
j - i + 1 == k

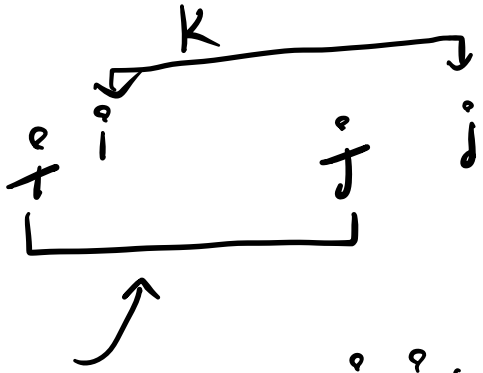
max_mEx = ∅∅4

=> Problems => Sliding window where k was not given



j-i+1
2-0+1=3





Easy

$j - i + 1 = k$

max
min
Sum
dis