Sliding Window

- Raghav Goel

Sliding Window

- Useful for array based problems - subarray
- When to use?
- Optimization Technique
- Use of 2 pointers.

Sliding window of fixed size

- The size of window always remains the same.
- Adding and removing an element from the window can be done efficiently.

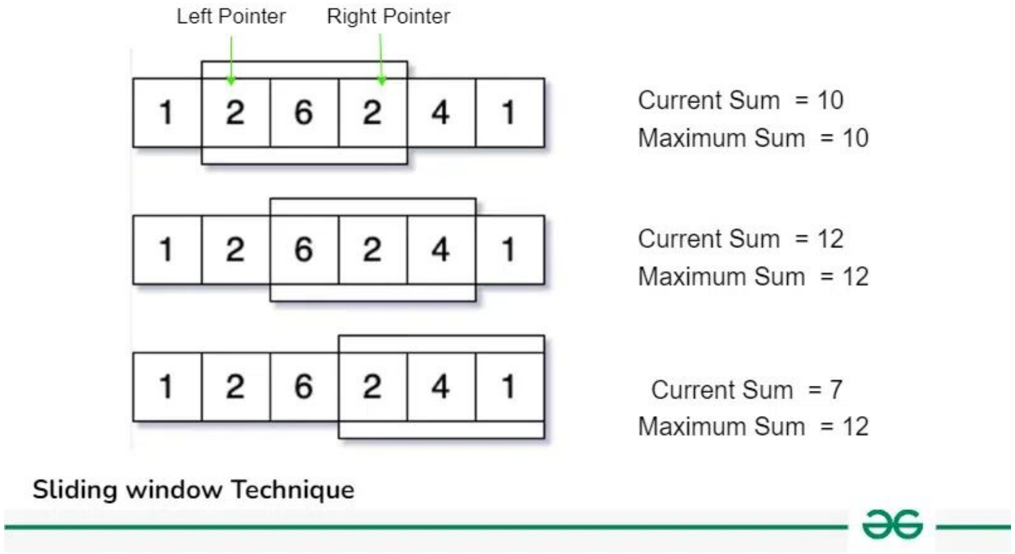
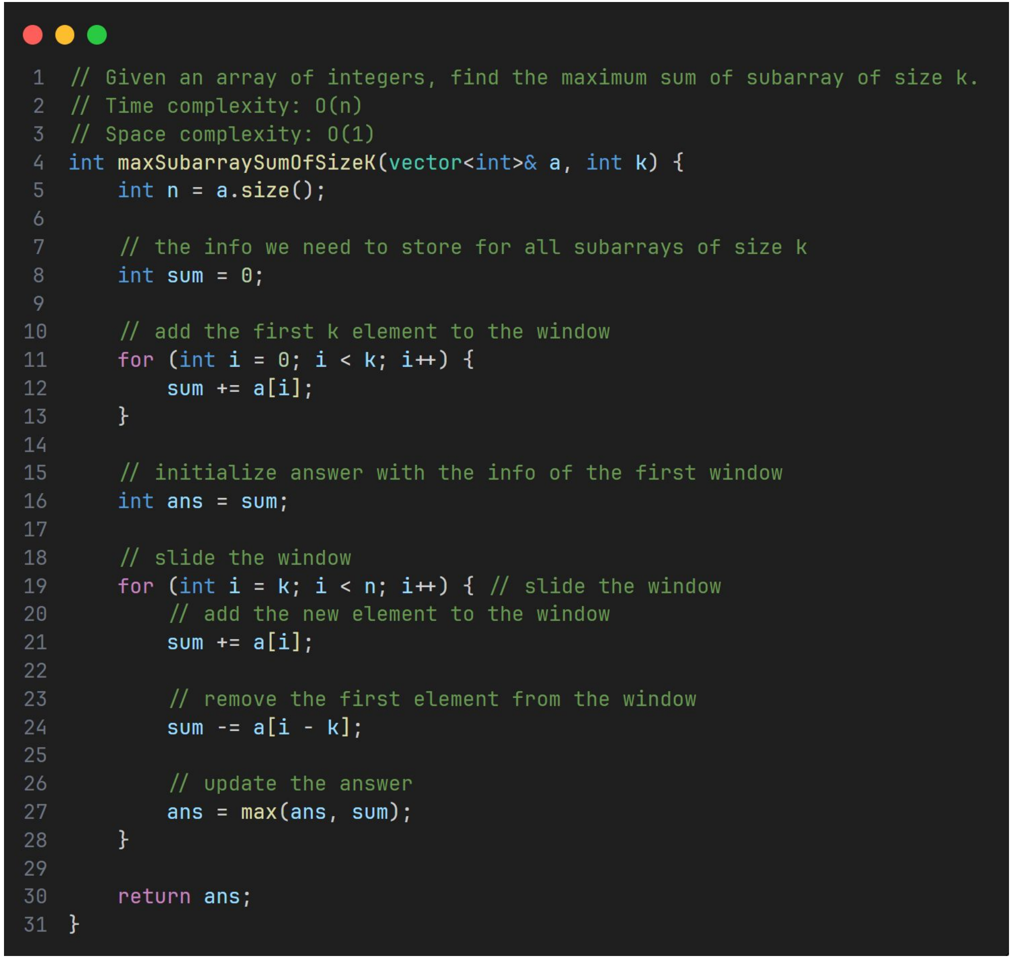


Image Credits - geeksforgeeks

Code

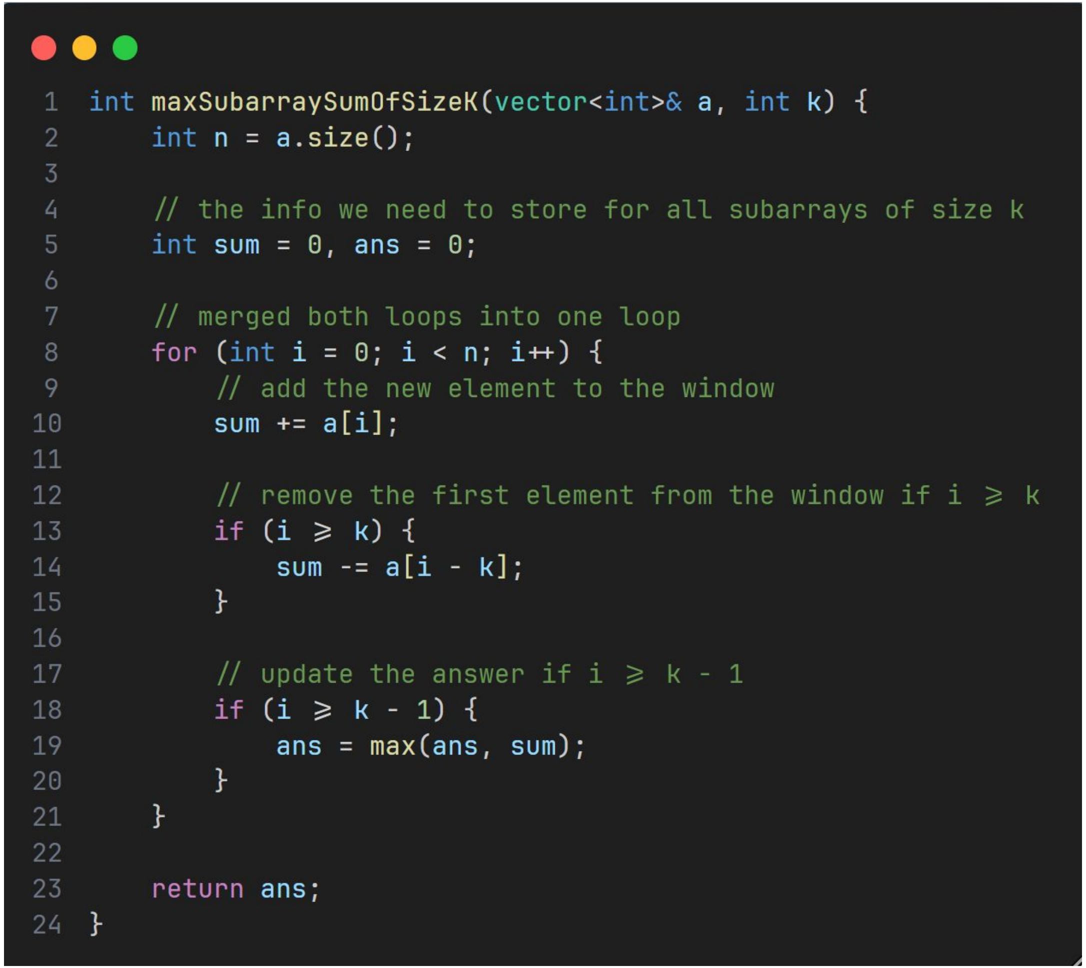
1. Decide what information you want to store for a window
2. Create first window of size
3. Initialize the answer using the information from the first window
4. Slide the window by adding a new element and removing the first element and update the answer in each iteration.



```cpp
// Given an array of integers, find the maximum sum of subarray of size k.
// Time complexity: O(n)
// Space complexity: O(1)
int maxSubarraySumOfSizeK(vector<int>& a, int k) {
    int n = a.size();

    // the info we need to store for all subarrays of size k
    int sum = 0;

    // add the first k element to the window
    for (int i = 0; i < k; i++) {
        sum += a[i];
    }

    // initialize answer with the info of the first window
    int ans = sum;

    // slide the window
    for (int i = k; i < n; i++) { // slide the window
        // add the new element to the window
        sum += a[i];

        // remove the first element from the window
        sum -= a[i - k];

        // update the answer
        ans = max(ans, sum);
    }

    return ans;
}
```

Code

Merged the for loops to make the code less repetitive and shorter



```cpp
int maxSubarraySumOfSizeK(vector<int>& a, int k) {
    int n = a.size();

    // the info we need to store for all subarrays of size k
    int sum = 0, ans = 0;

    // merged both loops into one loop
    for (int i = 0; i < n; i++) {
        // add the new element to the window
        sum += a[i];

        // remove the first element from the window if i >= k
        if (i >= k) {
            sum -= a[i - k];
        }

        // update the answer if i >= k - 1
        if (i >= k - 1) {
            ans = max(ans, sum);
        }
    }

    return ans;
}
```

Problems

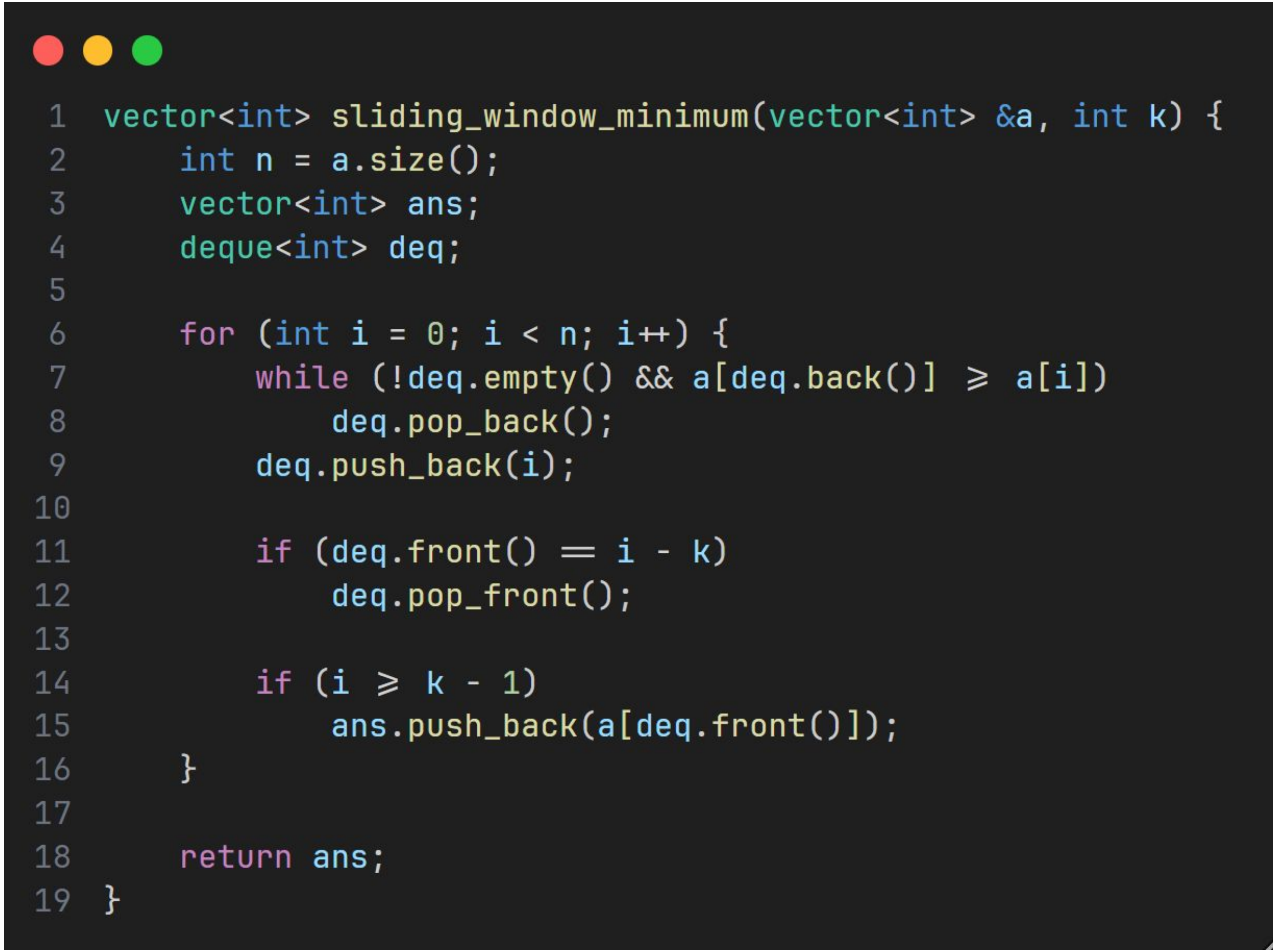
- Max. subarray sum of all subarrays with size k
- Distinct number of elements in all subarrays of size k
- First negative element in all subarrays of length k
- Max. subarray sum of subarrays with distinct elements of size k
- Maximum elements of all subarrays of size k

Queue / Deque Optimizations

- The elements which are added first in the window will be removed first, so sometimes we can optimize our codes by using queue / deque instead of sets.
- Example -> if we have to find the first negative element for all subarrays, we can store the indices of negative elements in a queue and the index of first negative element of window will be present at the front of queue for all subarrays.
  (generally we don’t require to optimize this much, but in interviews it will leave a very good impression)

Code

Sliding Window Minimum in O(N)



```cpp
vector<int> sliding_window_minimum(vector<int> &a, int k) {
    int n = a.size();
    vector<int> ans;
    deque<int> deq;

    for (int i = 0; i < n; i++) {
        while (!deq.empty() && a[deq.back()] >= a[i])
            deq.pop_back();
        deq.push_back(i);

        if (deq.front() == i - k)
            deq.pop_front();

        if (i >= k - 1)
            ans.push_back(a[deq.front()]);
    }

    return ans;
}
```

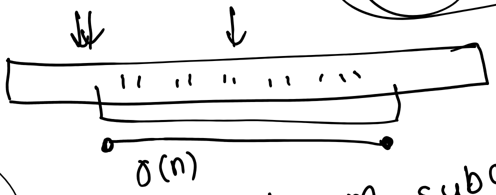
Resources

- Monotonic Queue (CP Algorithms) (good to know)
- Sliding Window (USACO)

Sliding Window

${}^nC_2 + n$

array



maximum subarray sum

$$\frac{n(n+1)}{2}$$

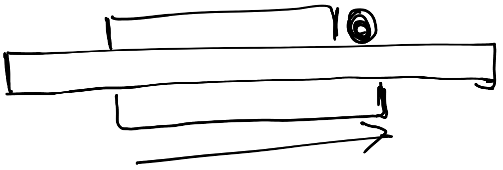
$O(n^2) * \cancel{O(n)}$

$\cancel{O(n^3)} \rightarrow O(n^2)$ ans

$1 \quad -1 \quad \overbrace{5 \quad -4 \quad 5}^{6}$

Kadane algorithm

previous sum + current element



maximum subarray sum of any subarray of size k

k=3
[1,-2,3,4,-7,8]

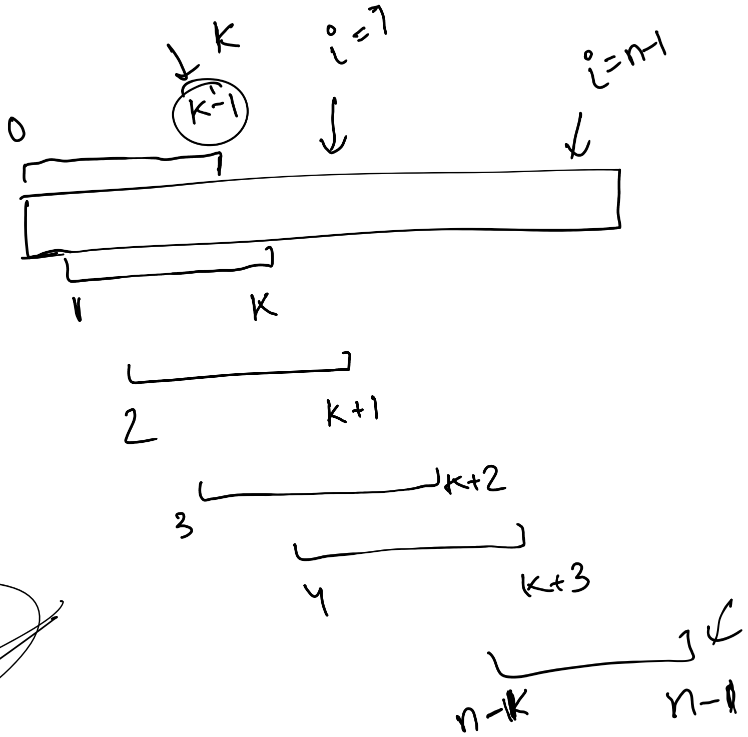
$O(n^2)$ subarrays of any size

? subarrays of size K

(n-k ~~n/k~~ k+1)

~~(n-~~
~~n-K~~
n-K+1

? subarrays of size K

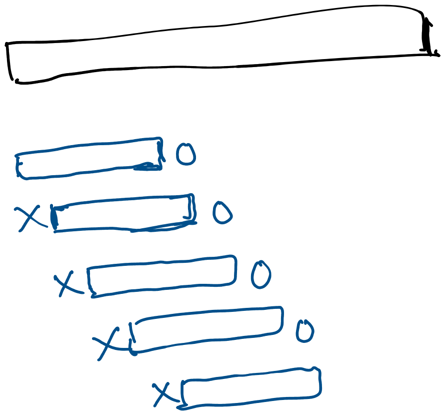


(n-1) - (K-1) + 1

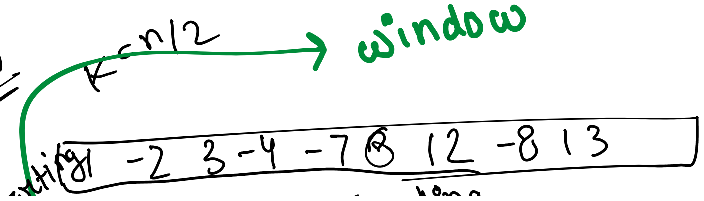
n-K+1

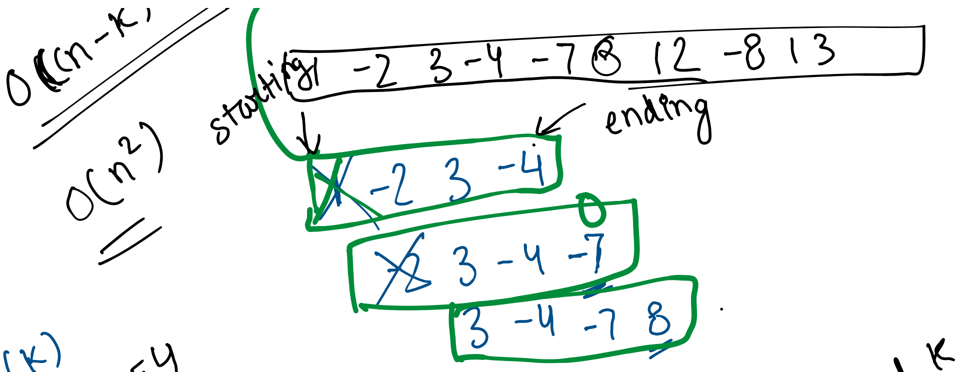
O(n)

Array



O((n-K)*K)





O(K) K=4

ans = -2

sum = ~~-2~~ -2 ~~3~~ ~~-10~~

~~-8~~ 0

l and K

2 - pointers

$\boxed{K}$

starting

ending?

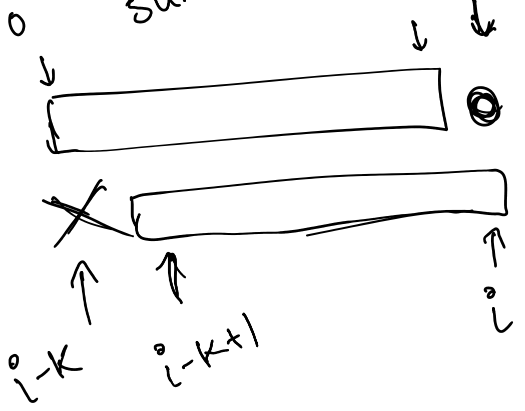
$\boxed{l}$ $\boxed{r} = l+K-1$

roll no. 8 to 21 => 21 - 8 + 1 = 14

14

1 to 21 => 21

ans = sum



i=K

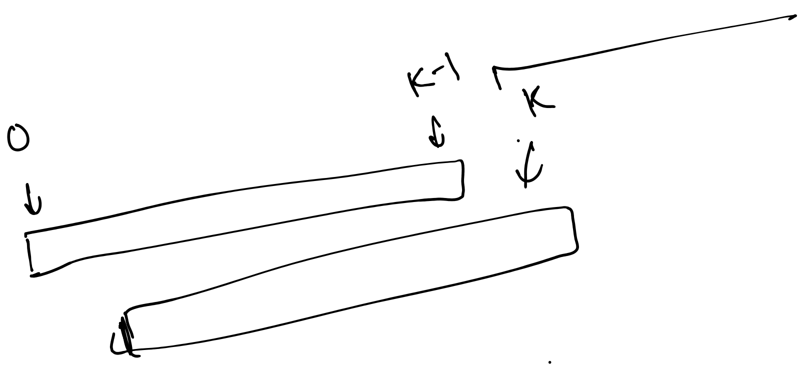
sum += a[i]

sum -= a[i-K]

ans = max(ans, sum)

- adding a[i] to the window
- removing a[i-K] from window

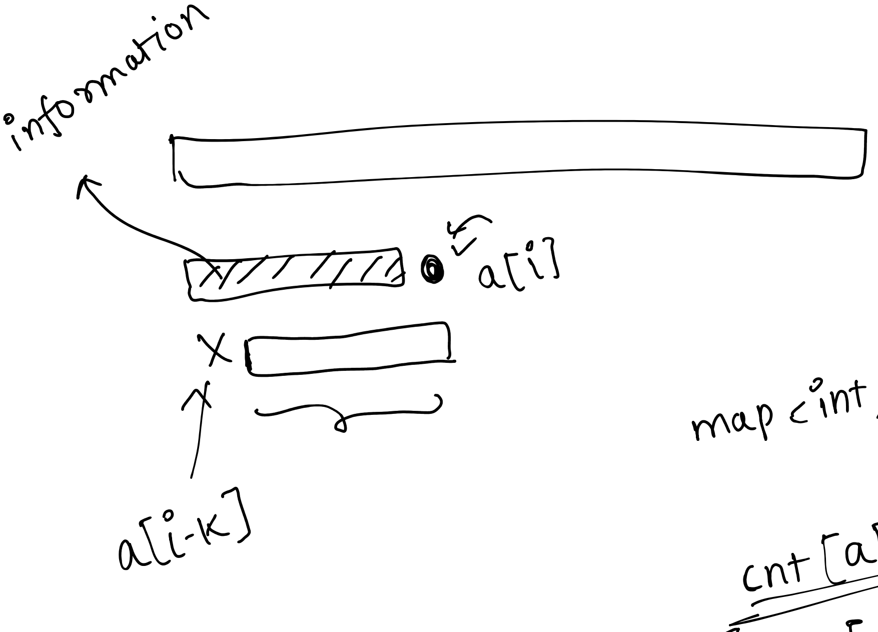
• updating answer



① maximum sum of any subarray of size K

② minimum sum of ....

③ maximum number of distinct elements in subarray of size K



map<int, int> cnt ;

```
cnt[a[i]]++;
i>=K    cnt[a[i-K]]--;
...ning    if ( cnt[a[i-K]] == ...
```

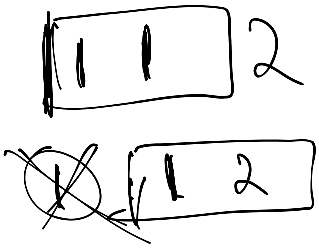
i ≥ k

erasing

```
{ cnt
  if (cnt[a[i-k]] == 0)
      cnt.erase(a[i-k])
```

updating the answer

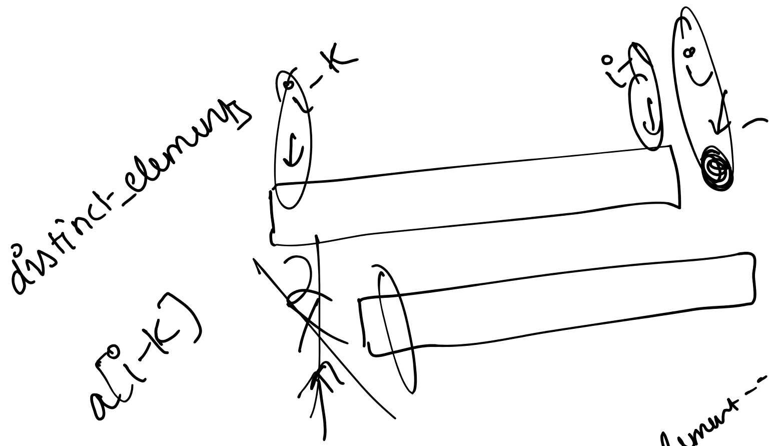
ans = max(ans, (int) cnt.size())



$\{\cancel{1}, 2\}$

r = 7

k=2

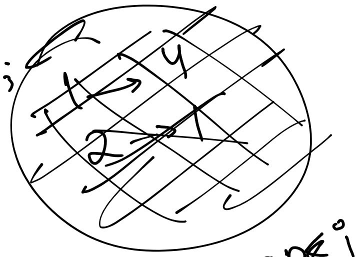


r = l+k-1

l = r-k+1

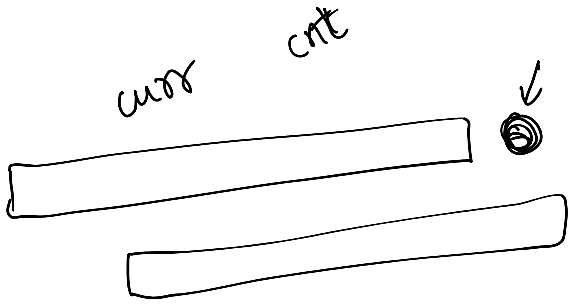
cnt[v]--;

if (cnt[v] == 0) distinct_element--;



map<int, int>

cnt [v]++



ans = $\cancel{0}$ 2

k = 3

=2

1 1 3

1 2 3

K =

curr = 2

1 → 2

2 → 0

3 → 1

1 2 1 1

1 1 3

K = 3

O(n*K) → intuitive ?

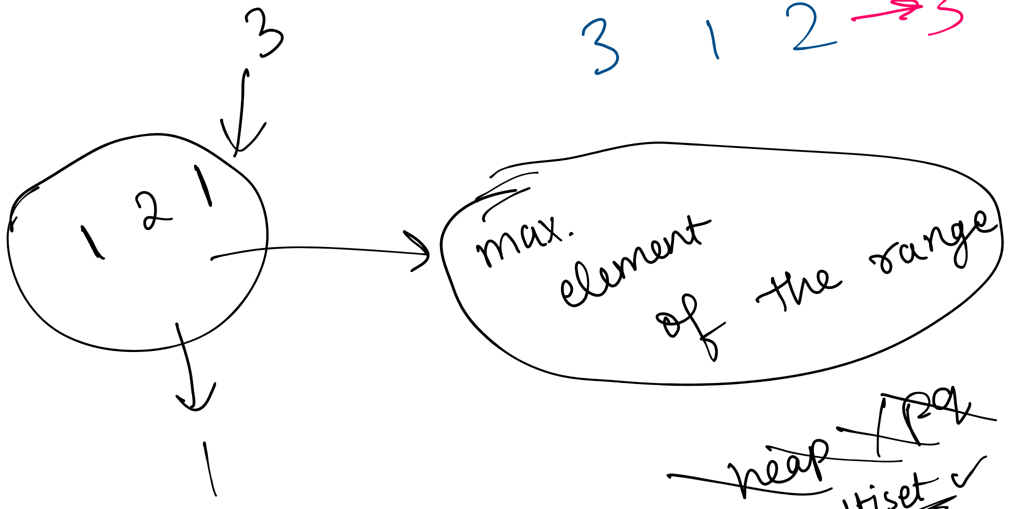
| 1 | 2 | 1 | 3 | 1 | 2 |

1 2 1 → 2

2 1 3 → 3

1 3 1 → 3

3 1 2 → 3



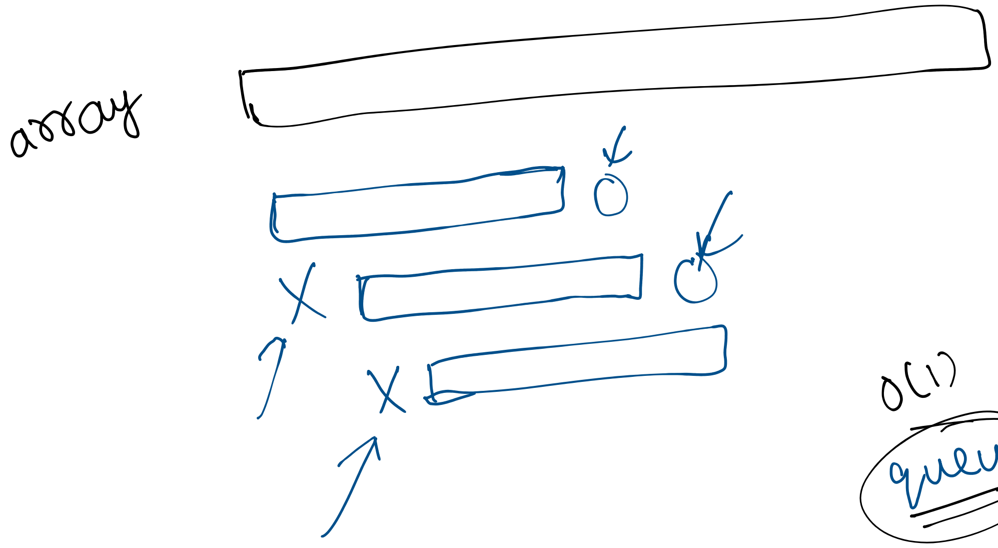
~~heap / pq~~

multiset ✓

deque → tricky

map ✓

~~set~~



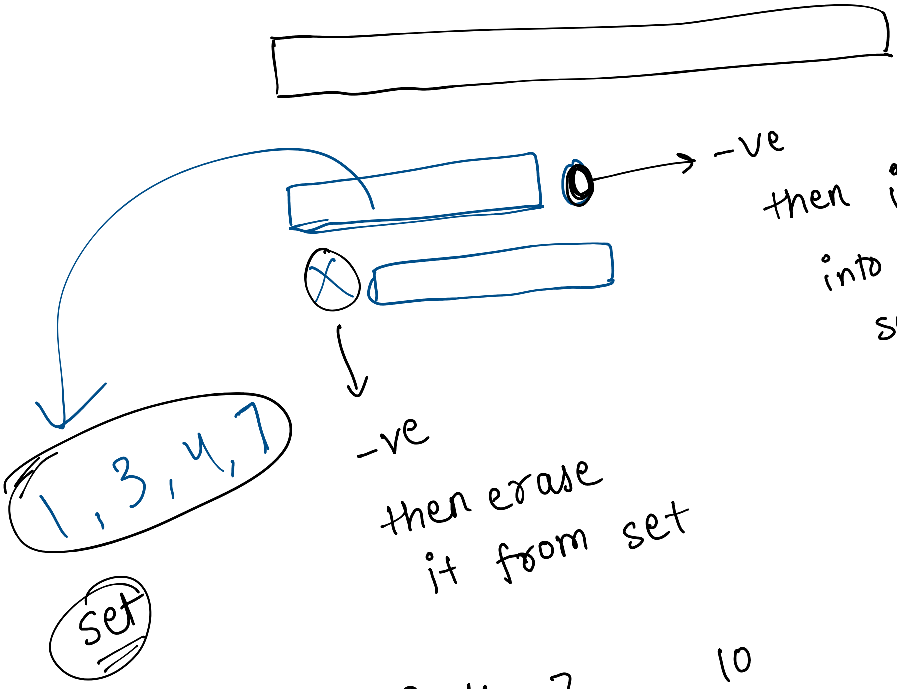
O(1)

queue

FIFO

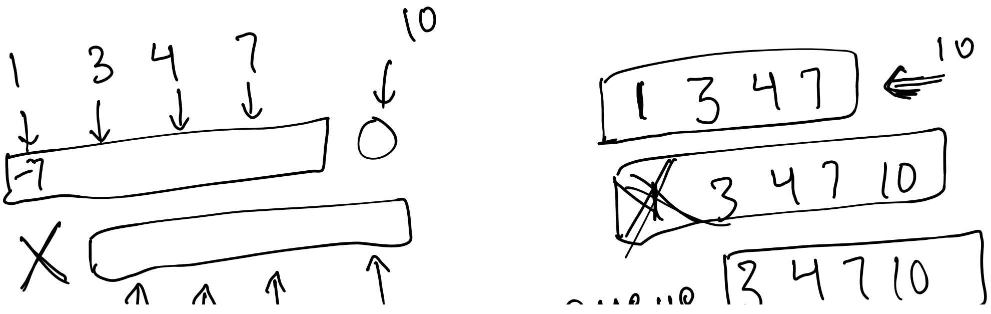
~~set~~ O(logn)

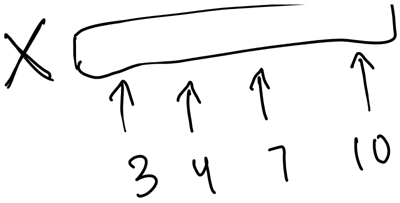
find first -ve element of all subarrays of size k

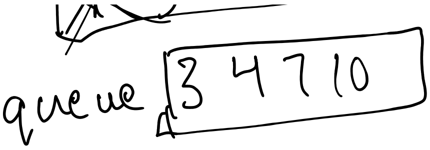


-ve then insert into the set

-ve then erase it from set







O(logN)

O(1)

O(NlogN)

O(N)

12 - 2 p.m.

Code - https://hastebin.com/share/uraruwekug.cpp