Problems On Queues

Created	@August 17, 2022 8:12 PM
Class	
Type	
Materials	
Reviewed	☐

Problem: First Non Repeating Character

You will be given stream of characters (i.e. we will be receiving multiple characters one by one). The moment we receive a new character, we need to return the first non repeating character at that point of time.

Example:

Character	Non Repeating char up till now
a	a
b	a
c	a
a	b
d	b
b	c
c	d
d	-1
a	-1
e	e

Because we are given a stream of characters, that means we have to answer immediately the moment we receive a character. If we have no non repeating character we print -1.

In the stream you might get max N characters and N ≤ 10^6.

Solution:

Brute Force:

In the brute force solution, the moment we receive a new character in the stream, we can try to check the whole set of characters received till now and then detect the first non repeating character by maintaining a frequency map kind of a thing.

Time: O(n^2)

How to optimise ?

We can have few observations:

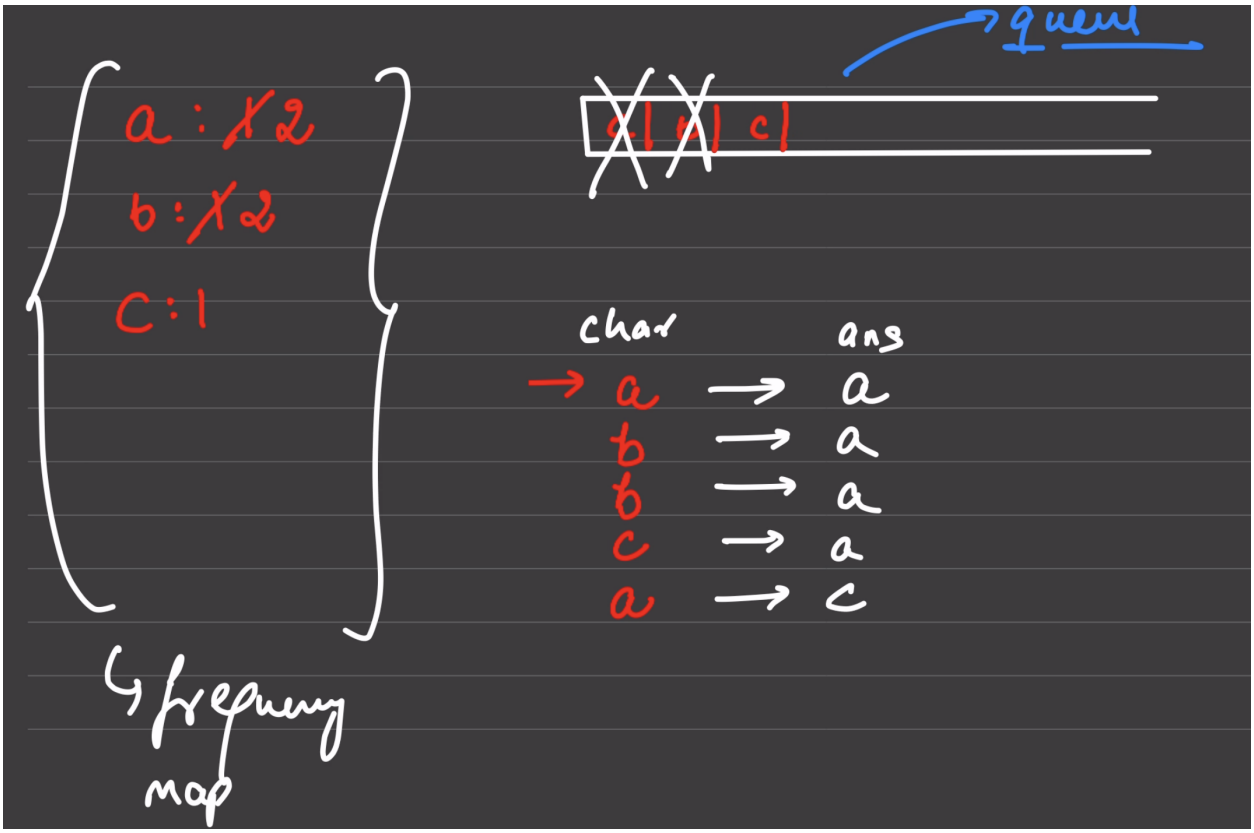
- The first character of the received is the first non repeating character.
- Even if after the first character we receive new unique characters then the new character can be a potential candidate later.
- So the first received character is the first candidate, then the second received character will be the second candidate (if unique) and so on.
- So we are getting an idea of first come first serve ? So may be queues can help.
- Apart from this, we are concerned about the occurrences of a character i.e. the frequency of a character, so it can be useful to prepare a frequency map also using objects or maps

So what we can do is, we can maintain a queue as well as a map.

The moment we receive a character, we can check the map if the character is received for the very first time or not. If the character is already received in the past, this character can never become the ans so we wont add it in the queue but increase the frequency so that we can detect it. Otherwise we add it in the queue and make an entry in the map.

Now we check the front of the queue, if the frequency of the character in the front of the queue is 1, then this front character is the answer, otherwise, we will keep on dequeuing

till the time we receive a character with frequency 1. If the queue gets empty we get the ans as -1.




```
function firstNonRepeatingChar(str) {
  /**
   * Time: O(n)
   * Space: O(1)
   */
    let qu = new Queue();
    let mp = {};
    for(let i = 0; i < str.length; i++) {
        const currChar = str[i];
        if(!mp[currChar]) {
      mp[currChar] = 1;
      qu.enqueue(currChar);
    } else {
      mp[currChar]++;
    }
    while(mp[qu.getFront()] > 1) {
      qu.dequeue();
    }
    if(qu.getFront() != undefined) {
      console.log(qu.getFront());
```

```
    } else {
      console.log(-1);
    }
    }
}
```

The time complexity will be `O(n)` and space will be constant because the frequency map will be only 26 entries and queue can also have at max 26 entries considering only 26 unique characters in the stream.

Problem 2:

Sliding Window Maximum - LeetCode

You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the

https://leetcode.com/problems/sliding-window-maximum/

Brute force:

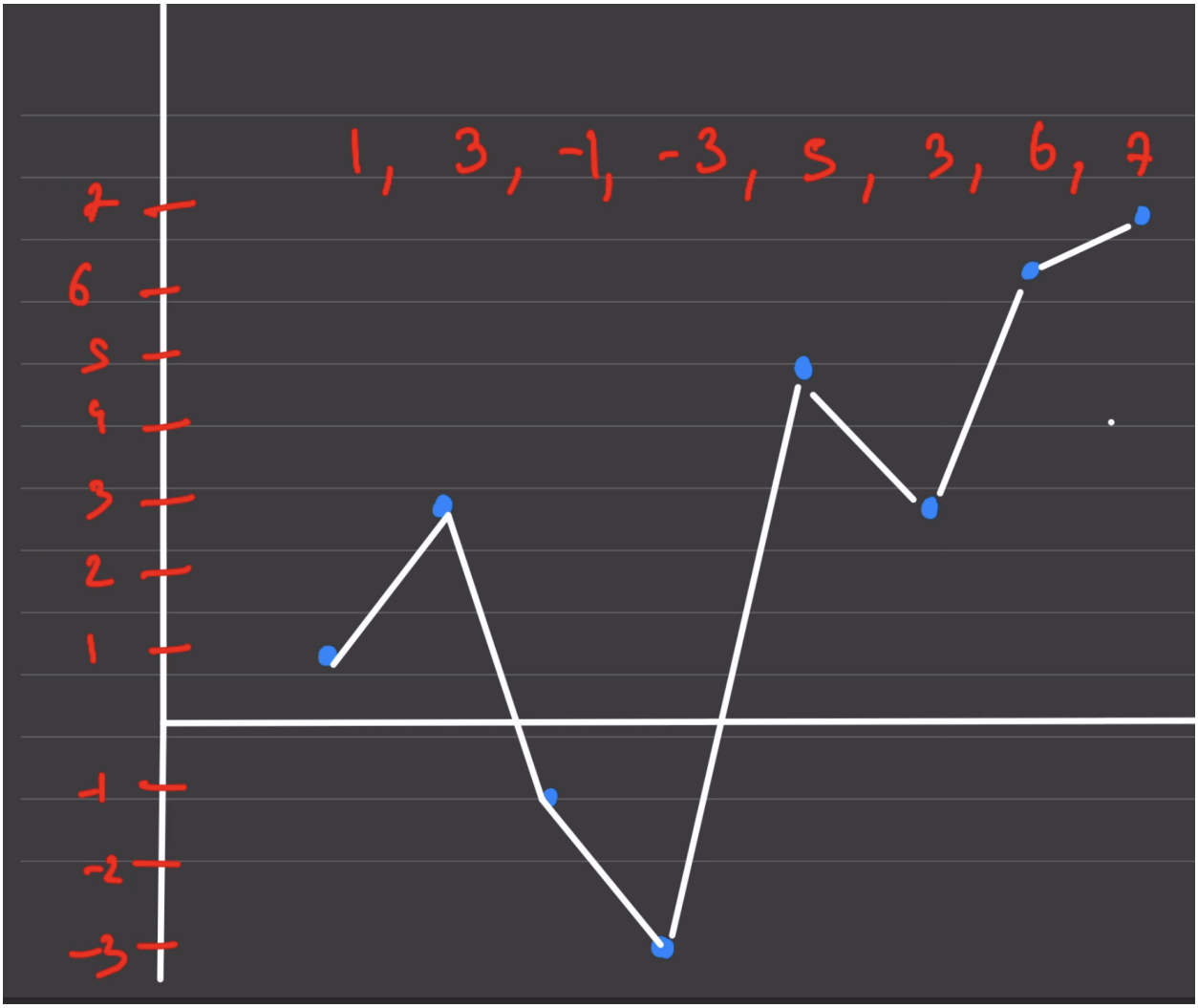
We can try to generate all possible subarrays of length K and then extract out the max of each subarray.

Time: O(n^2)

How can we optimise ?

`1,3,-1,4,5,3,-6,-7`

Hint: If we know the max of a current subarray, then the next subarray will introduce a new element, if this new element is less than the prev max, then we need to store this new element also, but the answer is still the prev max, whereas if the new element is greater than prev max, then we have a new ans and we don't need prev max, so we remove it and just store the new element.



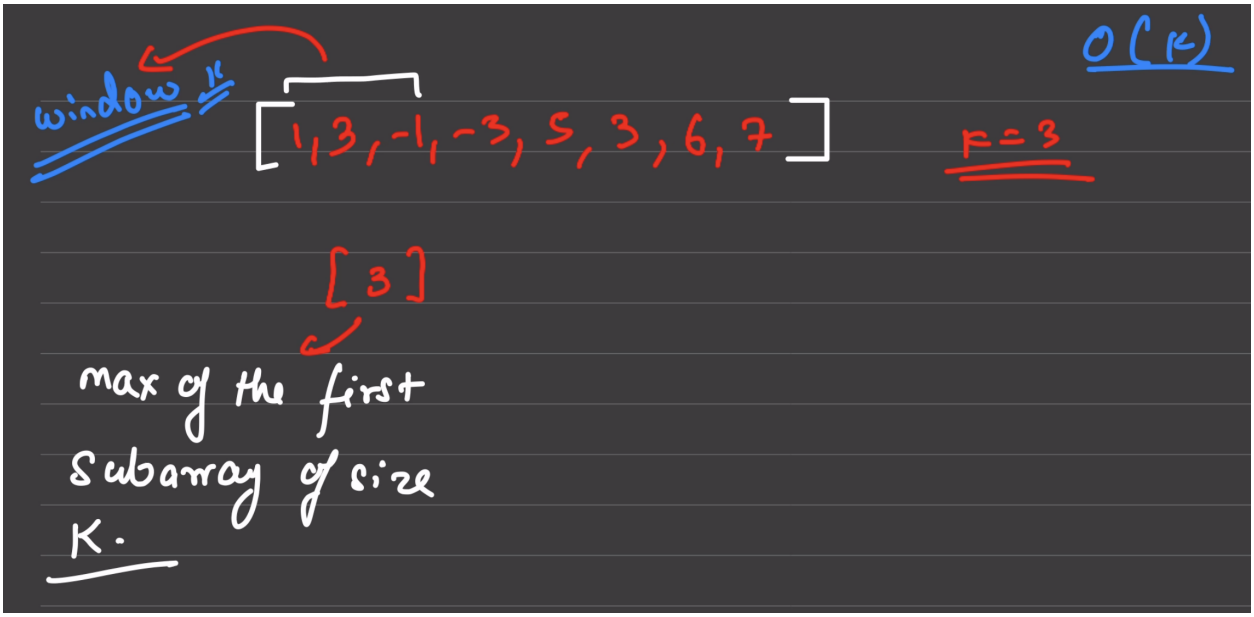
The above graph represents that, when we have smaller elements incoming in the new subarray, they can be a potential ans for some future subarrays, but the prev maximum will still be the maximum. Whereas if we have a new element which is greater than the previous candidates, then we don't need the prev candidates as the new element will be a better choice than them .

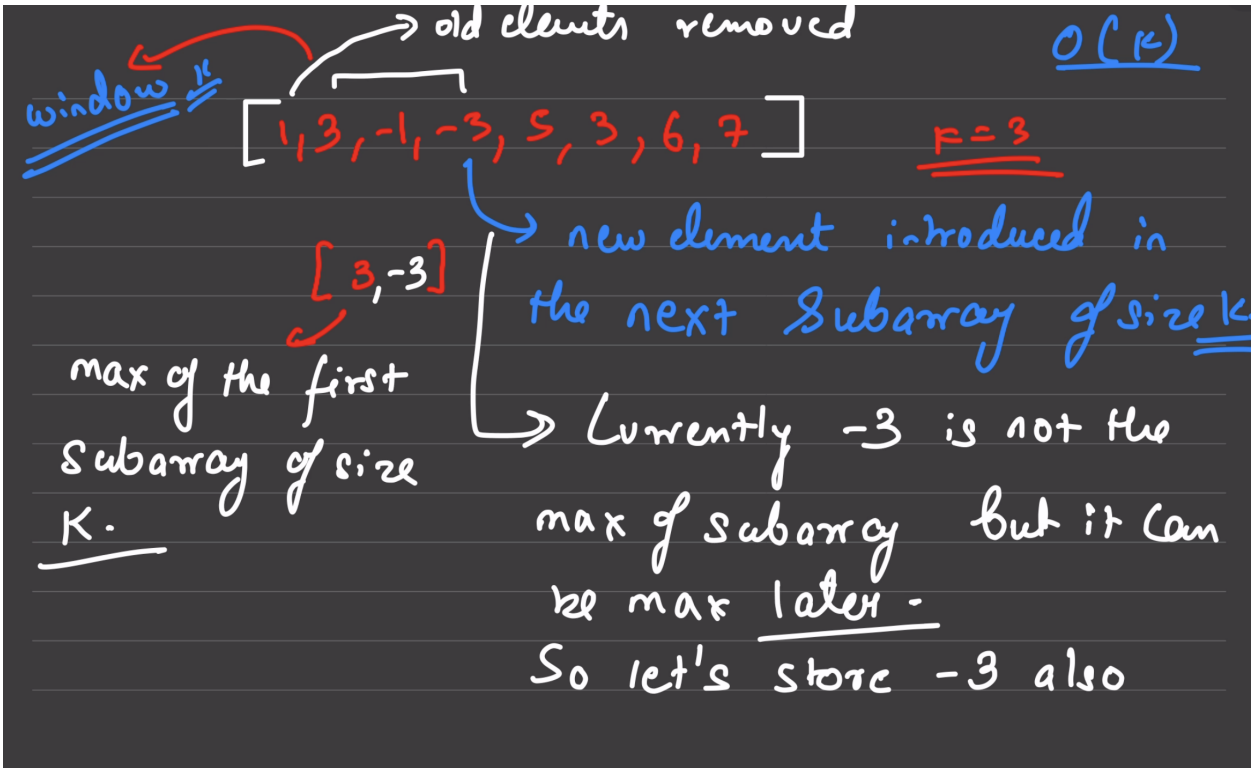
We can try to maintain a window of size k, if we are on a decreasing curve, then we will keep on adding the element. In this window the first element present will be the max. And if we receive an element which is greater than few of the previous elements then we can start removing them from the last.
So we need a structure, so that we can add elements from the back, remove them from

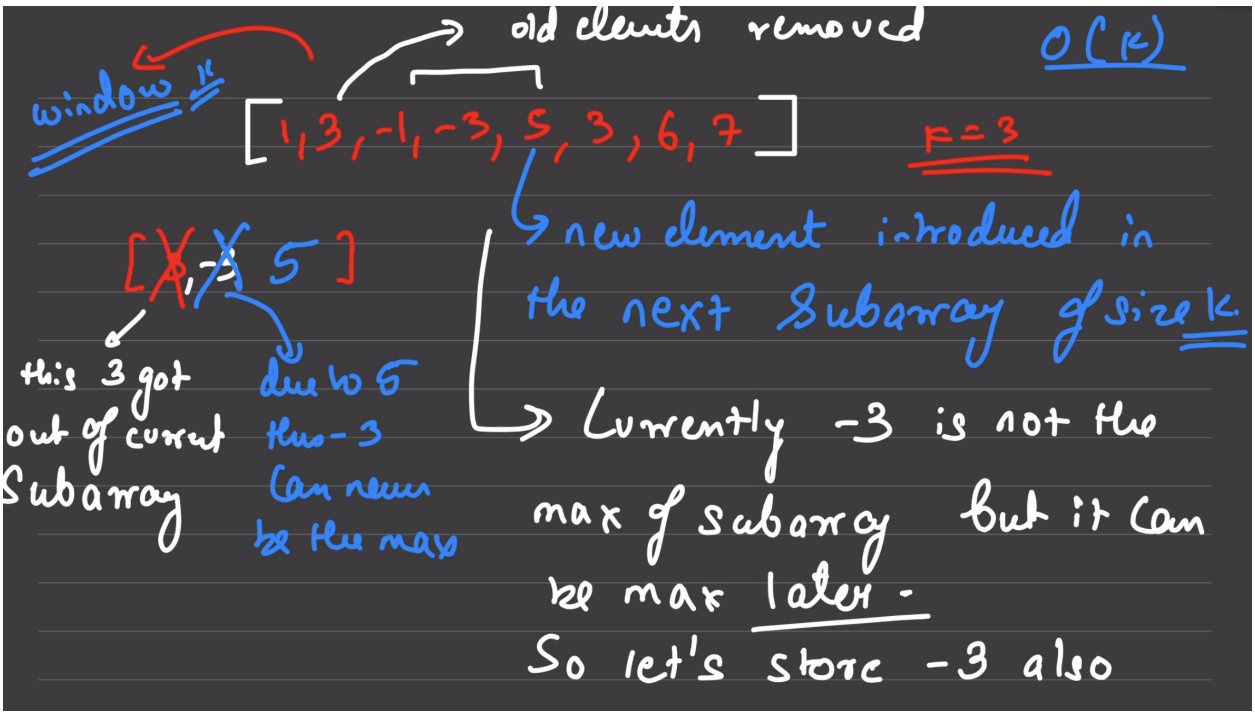
the back and remove elements from front also (when your current maximum gets out of the next subarray window)

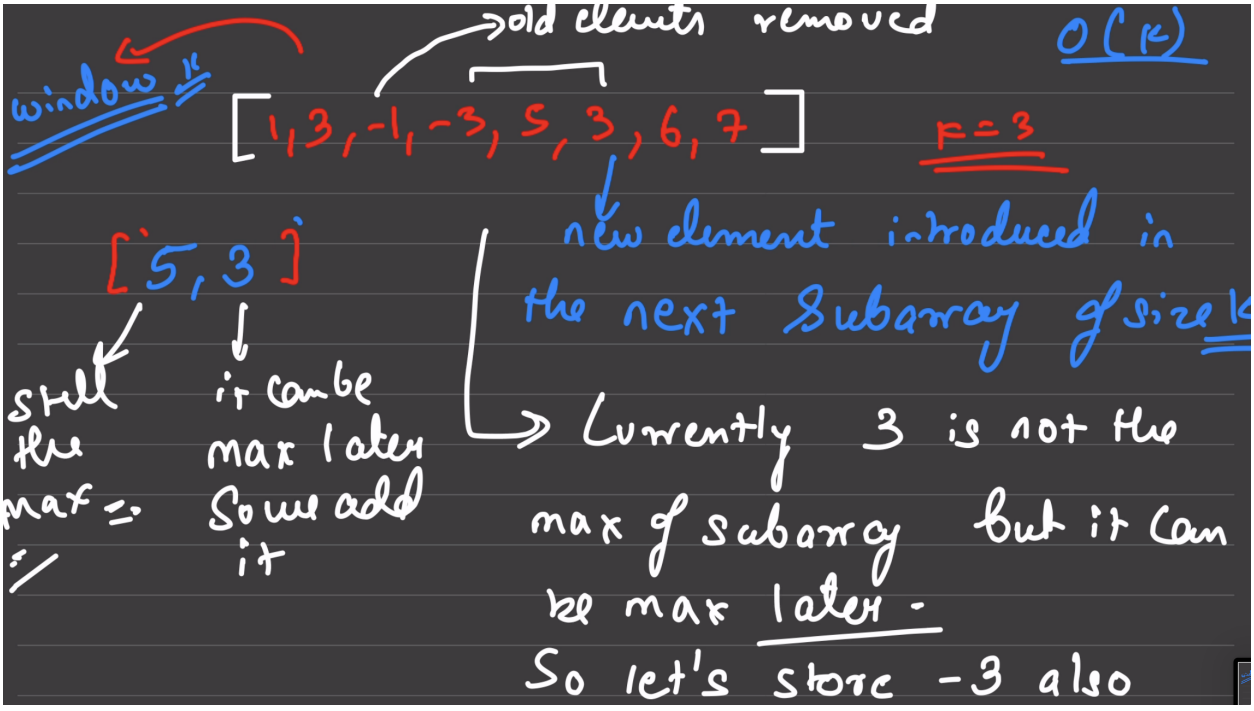
We can use a deque for this.

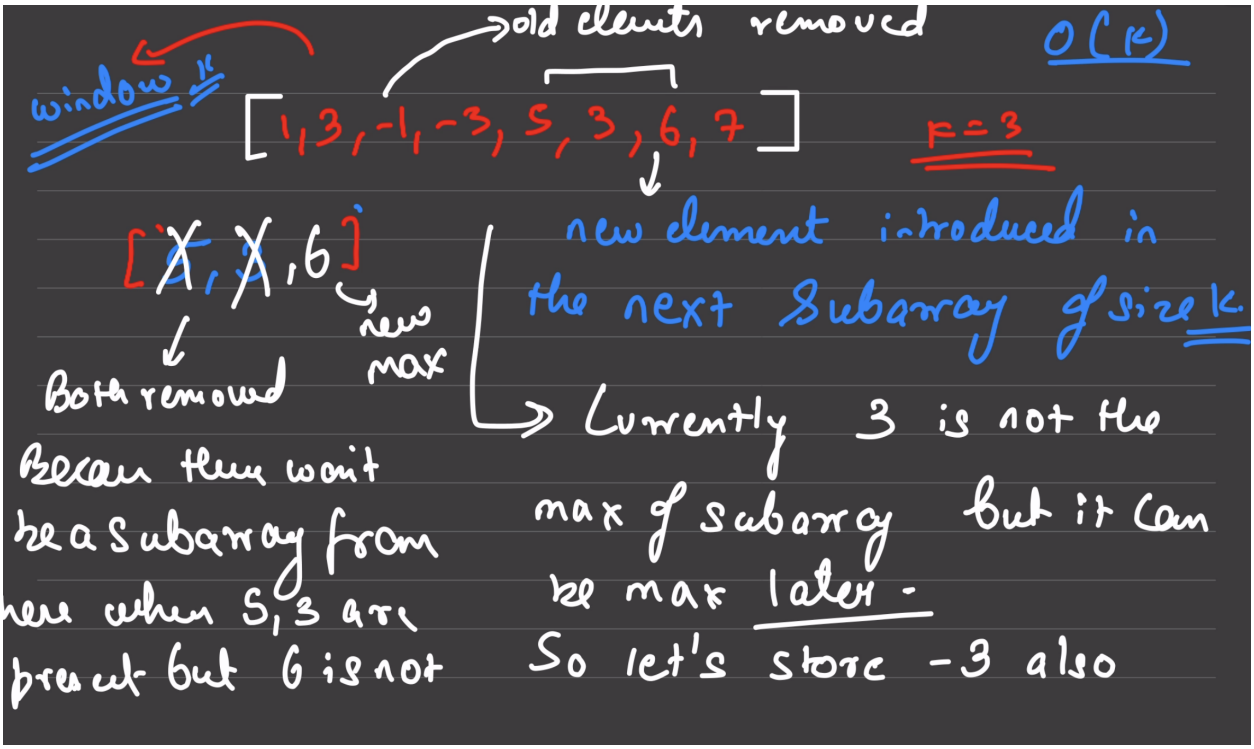
So in the deque if we have a decreasing curve, we will add all the elements from the back, and the element at front will be the max, but if we see an increasing curve, then from the back we need to remove all those candidates which dont make sense anymore.




old elements removed
O(K)
window K
[1,3,-1,-3,5,3,6,7]
K=3
new element introduced in the next Subarray of size K
[3,-3]
max of the first Subarray of size K.
Currently -3 is not the max of subarray but it can be max later.
So let's store -3 also


old elements removed
O(K)
window K
[1,3,-1,-3,5,3,6,7]
K=3
new element introduced in the next Subarray of size K.
[X X 5]
this 3 got out of current Subarray
due to 5 thus -3 can never be the max
Currently -3 is not the max of subarray but it can be max later.
So let's store -3 also


window
old elements removed
O(k)
[1,3,-1,-3,5,3,6,7]
k=3
[5,3]
still the max =
it can be max later So we add it
new element introduced in the next subarray of size k
Currently 3 is not the max of subarray but it can be max later -
So let's store -3 also


window
old elements removed
O(k)
[1,3,-1,-3,5,3,6,7]
k=3
[X,X,6]
new max
Both removed
Because there won't be a subarray from here when 5,3 are present but 6 is not
new element introduced in the next subarray of size k
Currently 3 is not the max of subarray but it can be max later -
So let's store -3 also