SQP11 Sliding Window Maximum

Sunday, December 4, 2022 5:28 PM

(L10) Sliding Window Maximum

Problem Statement that you are the given integer with (k) integer and your task is finding the maximum element of every window that size is (k) that is the question say

example

```
Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
Output: [3,3,5,5,6,7]
Explanation:
Window position                Max
---------------                -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

(7) → that is my answer

Thought Process

→ Always Start with Brute force Approach in Interview then go with the Optimized

	1	3	-1	-3	5	3	6	7
index =	0	1	2	3	4	5	6	7

k = 3

→ Now can I say that I start with Iteration from 0 and finding the all the possible Subarray with the length of K

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

1, 3, -1 = 3
3, -1, -3 = 3
-1, -3, 5 = 5
-3, 5, 3 = 5
5, 3, 6 = 6
3, 6, 7 = (7) → Ans (Maximum Answer)

How is look like that My Code

```
for(i=0 to len(arr) - k){
    Max = arr[i]
    for(j=i to i+k-1){
        Max = max(Max, arr[j])
    }
    return Max
}
```

→ that is the my possible answer code

Time Complexity = O(N x K) ≈ O(N²)

→ that is Quadratic time so Interviewer asked to Optimized that Approach

→ That is Quadratic time So Interviewer asked to Optimized that Approach

Optimized Approach

Index =	0	1	2	3	4	5	6	7
	1	3	-1	-3	5	3	6	7

K=3

→ Now here again we used the concept of Next Greatest element that we learned previously

→ In this Concept we Store the element into the increasing Fashion ↑

→ In this problem we store the element into the decreasing Order Fashion ↓

→ We required the deque data structure

→ We start Iteration from index = 0

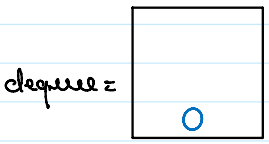
→ And maintain the deque property that is decreasing order ↓

→ and put that index into the deque

Index =	0	1	2	3	4	5	6	7
	1	3	-1	-3	5	3	6	7

K=3

→ i



Index Values

1 ↓

i = 0

→ Now check if deque is Empty then simply put the i into the deque

i = 1

→ Again Same Approach follow and maintain the property of deque and check

→ And check if deque is Not Empty So

→ existing element is > then arr[i] ✓

✓ → if that condition arr True then Simply put that index into the deque

X → otherwise Removed that element from the deque that is < then arr[i]

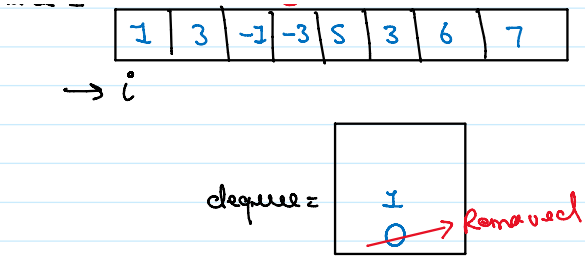
arr[i] = 3

→ Index = 1 < 3 ✓ (Removed from deque) → arr[i] < deque.front() ✓

Index =	0	1	2	3	4	5	6	7
	1	3	-1	-3	5	3	6	7

K=3

→ i

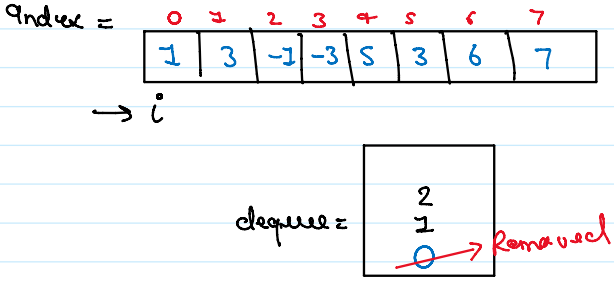


$i = 2$

→ Again same Approach follows and check if the deque follows the property or Not

arr[i] = -1 < deque.front ✓

↳ So simply append that i into the deque



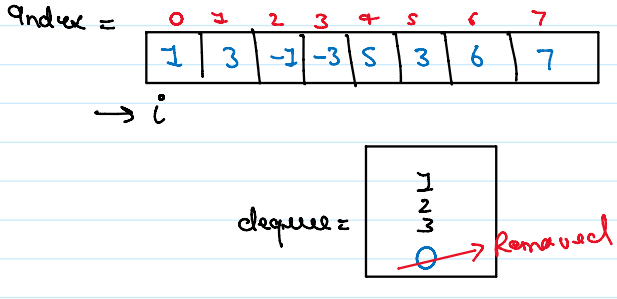
0 to 2
Max = 3

$i = 3$

→ Again same Approach follows and check if the deque follows the property follows or Not

arr[i] = -3 < deque.front ✓

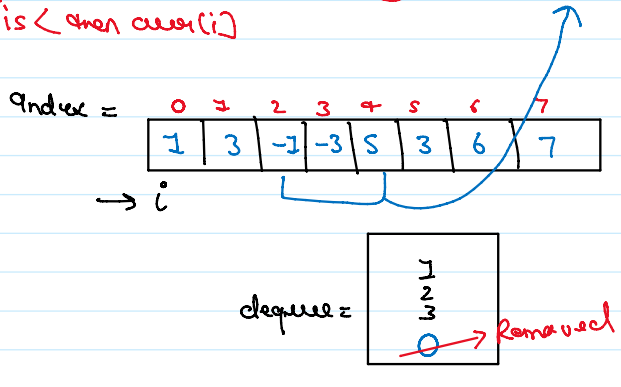
↳ So Simply append that i into the deque



1 to 3
Max = 3

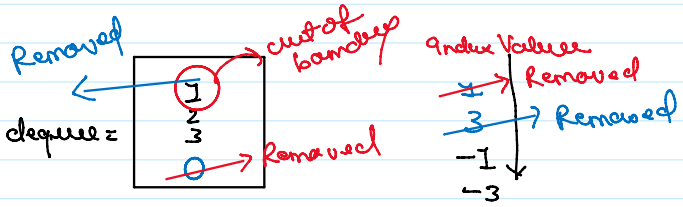
$i = 4$

Once we complete the one Half of the (K) that Means Removed all the element from the deque that is < than arr[i]



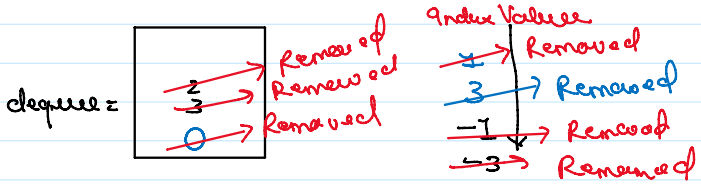
3 to 4
Max = 5

→ and check if any thing that is smaller than my boundary so simply pop from the front

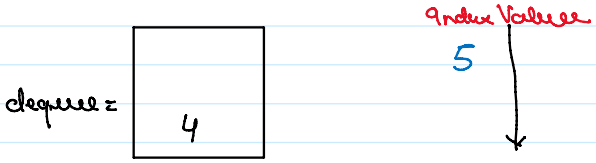


My boundy is = [2,4]

→ after removing my out of boundary element check from the back ← if any thing that is < than arr[i] in deque so simply Removed all element from the deque



→ After removed All the element that is < than arr[i] Simply push the i into the deque

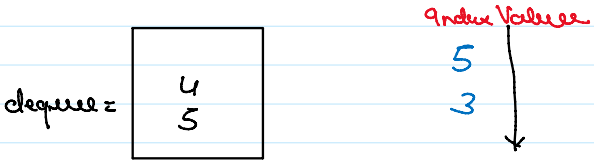


i = 5

→ Again same Approach follow and check if the deque follows the property follow or Not

arr[i] = 3 < deque.front ✓

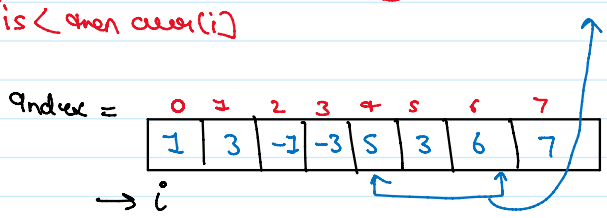
↳ So simply append that i into the deque



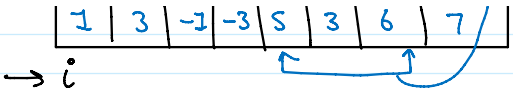
3 to 5
Max = 5

i = 6

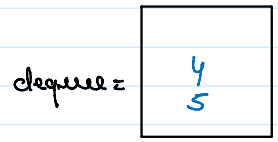
Once we complete the one half of the (k) that means Removed all the element from the deque that is < than arr[i]

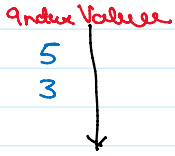


k = 3



(K=3)

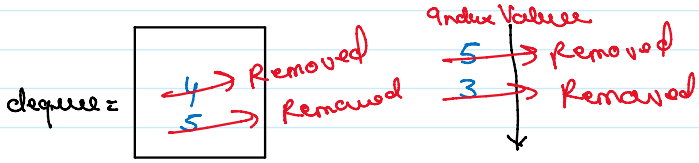




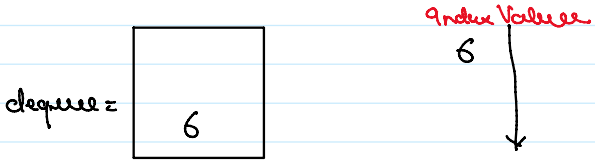
4 to 6
Max = 6

My boundary = [4, 6]

→ Now No any element next out of bound on the deque
→ check from the back ← if any element next is < than curr(i) into the deque so Removed all

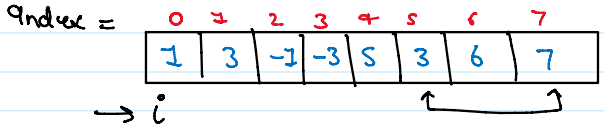


→ either Removed from back put i into the deque



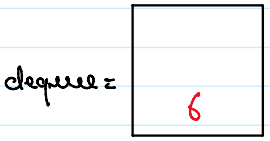
i = 7

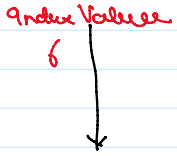
Once we complet the one half of the (K) that means Removed all the element from the deque next is < than curr(i)



(K=3)

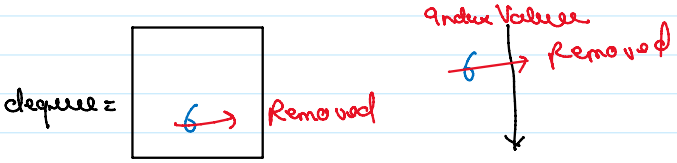
5 to 7
Max = 7



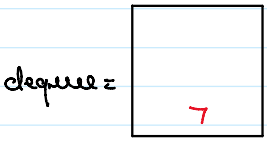


My boundary = [5, 7]

→ Now No any element next out of bound on the deque ✓
→ check from the back ← if any element next is < than curr(i) into the deque so Removed all



→ Either Removed from back put i into the dequeue



index Values
7 → that is the my last Maximum under the window of (k)

Maximum of all window = 3, 3, 5, 5, 6, 7

Time Complexity = O(N) + O(N) ≈ O(N)

→ Two linear Operation
→ Two Removing one out of bound index and that all element that is < then arr[i] from dequeue

→ that is Amotized Time Complexity

Space Complexity = O(k)

→ at one time only we stored one at max k element into the dequeue

#Implementation Python

```
# lc: https://leetcode.com/problems/sliding-window-maximum/submissions/
# gfg: https://practice.geeksforgeeks.org/problems/maximum-of-all-subarrays-of-size-k3101/1

import collections
class Solution:
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
        res = []
        window = collections.deque()
        for i, num in enumerate(nums):

            while window and num >= nums[window[-1]]:
                window.pop()

            window.append(i)

            if i >= k - 1:
                res.append(nums[window[0]])

            if i - window[0] + 1 == k:
                window.popleft()

        return res
```

#Implementation Java

```
import java.util.*;
public class L10_Sliding_Window_Maximum {
    public static void main(String[] args) {
        System.out.println("L10
_Sliding_Window_Maximum");
    }
}

class OptimizedSolution {
    public int[] maxSlidingWindow(int[] nums, int k) {

        int n = nums.length;
        int[] r = new int[n - k + 1];
        int ri = 0;
        // store index

        Deque < Integer > q = new ArrayDeque < > ();
        for (int i = 0; i < nums.length; i++) {

            // remove numbers out of range k
            if (!q.isEmpty() && q.peek() == i - k) {
                q.poll();
            }
            // remove smaller numbers in k range as they are
```

```
window.popleft()

return res
```

```
            q.poll();
        }
        // remove smaller numbers in k range as they are useless
        while (!q.isEmpty() && nums[q.peekLast()] < nums[i]) {
            q.pollLast();
        }

        q.offer(i);

        if (i >= k - 1) {
            r[ri++] = nums[q.peek()];
        }
    }
    return r;


  }
}
```