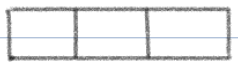
Maximum sliding window: Given an array of integers arr and a sliding window of size k (k<=n), find maximum sliding window; sliding window starts from 0th index of the array and moves one step to the right side at each iteration.

arr = [2 4 1 3 0 12]

k = 3 (size of the window) 

Question: How many window can we have with an array of size n and window size of k?

arr = [2 4 1 3 0 12]; n= 6; k= 3;

Iteration	sliding window
1	arr = [**2 4 1** 3 0 12]
2	arr = [2 **4 1 3** 0 12]
3	arr = [2 4 **1 3 0** 12]
4	arr = [2 4 1 **3 0 12**]
5	arr = [2 4 1 3 **0 12**] out of index

It can observed that with n= 6 and k= 3, we are having 4 (n−k+1) sliding windows.

Algorithm:

keep one reference index

Use one inner loop

Compare the reference node to each internal loop element which has the size of one window

append the largest to the sliding window.

Increment the outer loop reference by one after each complete inner loop completion.

```
def maxwindow (arr, n, k):
    window =[ ]
    for i in range( n -k+1):
        ref = arr[i]
        for j in range(i+1, i+k):
            ans = max( ref, arr [j])
        window. append(ans)
    return ans
```

Analysis: For outer loop, we do at most (n-k+1) operations and for inner loop (i+k) operation, Hence the time complexity can be computed as follow.

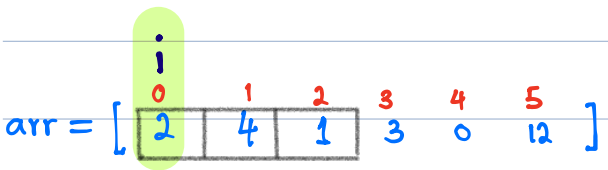
O(outer loop) * O(inner loop)

$\Rightarrow$ O(n-k+1) * O(i+k) $\Rightarrow$ Ignoring lower order terms

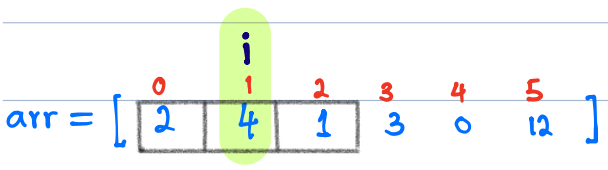
$\Rightarrow$ O(n)*(k)

optimal solution using dequeue:

arr = [2 4 1 3 0 12]; n = 6; k = 3;



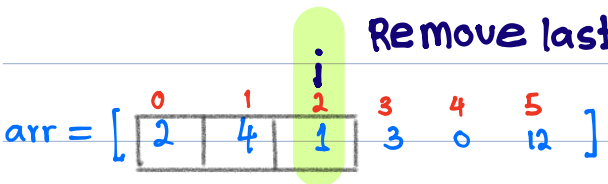
2



~~2~~ 4

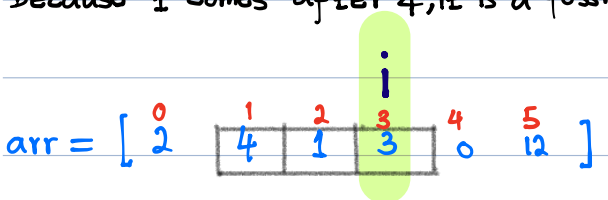
current element >= last element in dequeue:

Remove last element From dequeue:



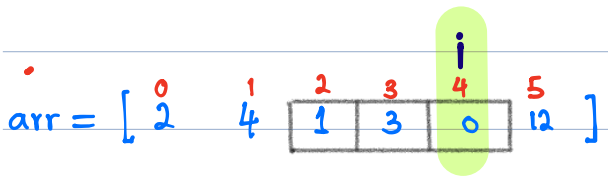
~~2~~ 4 1

because 1 comes after 4, it is a possibility to be our answer.



~~2~~ 4 ~~1~~ 3

1 comes before 3, and it is smaller than 3, cannot be our answer, because we want the largest in the range.



~~2~~ 4 ~~1~~ 3 0

because 0 comes after 3, it is a possibility to be our answer.

i

arr = [2 4 1 3 0 12] (indices 0 1 2 3 4 5)

~~2~~ 4 ~~1~~ 3 ~~0~~ 12

0 comes before 12, and it is smaller than 12; cannot be our answer, because we want the largest in the range.

our final answer is [4 3 12]

Analysis: for each element, we are doing addition, maybe removal or both.

in dequeue the above operations are done at O(1);

2 operations at worst case scenario for n elements

Hence, T.C = O(2 * N) = O(N)

```
def slidingwindow(arr, n, k):
    dq = dequeue()
    ans = [0] * (n-k+1)
    // solve for the first window
    For i in range(k):
        while len(dq)>0 and arr[i]>dq[-1]:
            dq.pop()
        dq.append(arr[i])
    ans[0] = deque[0]
// for remaining elements, insert ith
   and remove (i-k)th
    For i in range(k,n):
        while len(dq)>0 and arr[i]>dq[-1]:
            dq.pop()
        if arr[dq[0]] == arr[i-k]:
            dq.popleft()
        dq.append(arr[i])
        ans[i-k+1] = deque[0]

    return ans
```