```
1 2 3
```

What is a Deque (or double-ended queue) ?

The deque stands for Double Ended Queue. Deque is a linear data structure where the insertion and deletion operations are performed from both ends. Though the insertion and deletion in a deque can be performed on both ends, it does not follow the FIFO rule. The representation of a deque is given as follows -

Representation of deque

Fixed Window

In a fixed window we have a fixed length in which we have to traverse and find the solution. Now imagine if we have an array of 10 elements and a fixed window of size 3. Now we first check the first 3 elements, then we check 2,3,4 elements and so on. For this process we need two for loops.

This problem can be solved with the Sliding window technique.

Another example is when we need to check a certain property among all the sizes of an array then Sliding window technique also comes into picture.

Sliding window:

The Sliding window is a problem-solving technique of data structure and algorithm for problems that apply arrays or lists. These problems are painless to solve using a brute force approach in **O(n²) or O(n³)**. However, the **Sliding window** technique can reduce the time complexity to **O(n).**

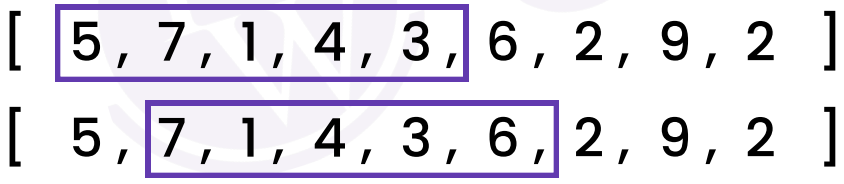

Figure 1: Sliding window technique to find the largest sum of 5 consecutive numbers.

The basic idea behind the sliding window technique is to transform two nested loops into a single loop.

Below are some fundamental clues to identify such kind of problem:

The problem will be based on an array, list or string type of data structure.

- It will ask to find subranges in that array or string and will have to give longest, shortest, or target values.
- Its concept is mainly based on ideas like the longest sequence or shortest sequence of something that satisfies a given condition perfectly.

Let's say that if you have an array like below:
[a b c d e f g h]

A sliding window of **size 3** would run over it like below:

```
[a b c]
  [b c d]
    [c d e]
      [d e f]
        [e f g]
          [f g h]
```

Q4 : 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 sliding window moves right by one position.

Return the max sliding window.

Example 1:
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
```

Example 2:
Input: nums = [1], k = 1
Output: [1]

Solution :

Code : LP_Code5.java

Output :

```
The desired output is :
5  5  5  9  9
```

Approach :

- We scan the array from 0 to n-1, keeping only useful elements in the deque. The algorithm is amortized O(n) as each element is put and polled once.
- At each i, we keep "promising" elements, which are potentially max numbers in window [i-(k-1),i] or any subsequent window. This means
- If an element in the deque is out of i-(k-1), we discard them. We just need to poll from the head, as we are using a deque and elements are ordered as the sequence in the array
- Now only those elements within [i-(k-1),i] are in the deque. We then discard elements smaller than a[i] from the tail. This is because if a[x] <a[i] and x<i, then a[x] has no chance to be the "max" in [i-(k-1),i], or any other

subsequent window: a[i] would always be a better candidate.

- As a result elements in the deque are ordered in both sequence in array and their value. At each step the head of the deque is the max element in [i-(k-1),i].

Time complexity : O(n) where n = size of the array
Space complexity : O(n)