Sliding Window Maximum

Find the maximum in a given window size for example window size =3

1 2 4 3 56

For every window size of 3 find the maximum element.

BRUTE FORCE :

Traverse every window using nested loops and find maximum for each window

Code :

```
#include <bits/stdc++.h>
vector<int> slidingWindowMaximum(vector<int> &nums, int &k)
{
    //    Write your code here.
    int mx;
    vector<int> ans;
    for(int i=0;i<=nums.size()-k;i++)
    {
        mx=INT_MIN;
        for(int j=i;j<i+k;j++)
        {
            mx=max(mx,nums[j]);
        }
        ans.push_back(mx);
    }
    return ans;
}
```

- Time Complexity : O(N*k)
- Space Complexity : O(1)

Optimized Approach :

Always maintain a descending order in the data structure. We need to insert smaller elements in the current window at the end of the deque and greater elements at the starting of deque. If a new window starts then just pop the elements that are out of window from the front. Deque will store only indexes. If deque contains smaller element and a greater element is found then we keep popping the elements less than it because

for all the windows for which this max element is a part the maximum element will be this element only.

Code :

```
#include <bits/stdc++.h>
vector<int> slidingWindowMaximum(vector<int> &nums, int &k)
{
    //    Write your code here.
    int mx;
    deque<int> dq;
    vector<int> ans;
    for(int i=0;i<nums.size();i++)
    {

          if (!dq.empty()&& dq.front() == i - k)
          {
            dq.pop_front();
          }
          while(!dq.empty()&&nums[dq.back()]<nums[i])
          {
              dq.pop_back();
          }
          dq.push_back(i);
        if(i>=k-1)
        {
            ans.push_back(nums[dq.front()]);
        }
    }

    return ans;
}
```

- Time Complexity : O(N)+O(N) one time traversed by deque for popping and once for outer for loop
- Space Complexity : O(k)