Maximum of minimum of window size

BRUTE FORCE :

Use sliding window on each window size and calculate maximum for it.

Code :

```
#include <bits/stdc++.h>
int slidingWindow(vector<int>& nums,int k)
{
    deque<int> dq;
    vector<int> minm;
    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)
        {
            minm.push_back(nums[dq.front()]);
        }
    }
    int mxm=*max_element(minm.begin(),minm.end());
    return mxm;
}
vector<int> maxMinWindow(vector<int> a, int n) {
    // Write your code here.
    vector<int> ans;
    for(int i=0;i<n;i++)
    {
        int res=slidingWindow(a,i+1);
        ans.push_back(res);
    }
    return ans;
}
```

- Time Complexity : O(n^2)
- Space Complexity : O(k)

OPTIMAL APPROACH :

Using concept of next smaller element on left and next smaller element on right.

For each element in the array we will determine that till which window size an element can be an answer. For example 10 20 30 10 50 70

10 can be a possible minimum for every window size i.e 1,2,3,4…7

20 can be possible minimum for size 2 at maximum.

and so on..

How to determine this ?

For calculating this we can find the next smaller element on left because sliding window is contiguious so if an element less than the current element is found then the lesser element will be the ans for that window so we will pop out elements from stack until elements greater than current element are present as in case of next smaller element. Similarly we will find on the right. For all the elements between this range will be the maximum window size for which the current element can be a possible answer.

After calculating the minimum we need to compute the maximum. For computing maximum we that if the current element is possible minimum for size k then it will be the minimum for k-1,k-1…1

So we start backwards and find whether elements at n-1 is greater than n-2 because if a number is maximum for window of size k-1 then it will definitely be maximum for window size k.

If we traverse the array from the beginning the problem is that for window size k+1 the element will not be the minimum so we traverse from the back.

At last we return the ans array

Code :

```
#include <bits/stdc++.h>
void nextSmallerLeft(vector<int> &nums,vector<int>& nsl)
{
    stack<int> s;
```

```
    for(int i=0;i<nums.size();i++)
    {
        if(s.empty())
        {
            nsl.push_back(-1);
        }
        else
        {
            while(!s.empty()&&nums[s.top()]>=nums[i])
            {
                s.pop();
            }
            if(s.empty())
            {
                nsl.push_back(-1);
            }
            else
            {
                nsl.push_back(s.top());

            }
        }
        s.push(i);
    }
}
void nextSmallerRight(vector<int> &nums,vector<int>& nsr)
{
    stack<int> s;
    for(int i=nums.size()-1;i>=0;i--)
    {
        if(s.empty())
        {
            nsr.push_back(nums.size());
        }
        else
        {
            while(!s.empty()&&nums[s.top()]>=nums[i])
            {
                s.pop();
            }
            if(s.empty())
            {
                nsr.push_back(nums.size());
            }
            else
            {
                nsr.push_back(s.top());

            }
        }
        s.push(i);
    }
    reverse(nsr.begin(),nsr.end());
}
```

```
vector<int> maxMinWindow(vector<int> a, int n) {
    // Write your code here.
    vector<int> minm(n,0),nsl,nsr;
    vector<int> ans(n,INT_MIN);
    nextSmallerLeft(a, nsl);
    nextSmallerRight(a, nsr);

    for(int i=0;i<n;i++)
    {
        minm[i]=abs(nsr[i]-nsl[i]-2);
    }
    for(int i=0;i<n;i++)
    {
        if(ans[minm[i]]==INT_MIN)
        {
            ans[minm[i]]=a[i];

        }
        else
        {
            if(a[i]>ans[minm[i]])
            {
                ans[minm[i]]=a[i];
            }
        }
    }
    for(int i=n-2;i>=0;i--)
    {
        if(ans[i]==INT_MIN)
        {
            ans[i]=ans[i+1];
        }
        else {
          ans[i] = max(ans[i], ans[i + 1]);
        }
    }
    return ans;
}
```

- Time Complexity : O(n)