Find maximum of minimum for every window size in a given array

Given an integer array of size n, find the maximum of the minimum's of every window size in the array. Note that window size varies from 1 to n.

Example:

```
Input:  arr[] = {10, 20, 30, 50, 10, 70, 30}
Output:         70, 30, 20, 10, 10, 10, 10

First element in output indicates maximum of minimums of all
windows of size 1.
Minimums of windows of size 1 are {10}, {20}, {30}, {50}, {10},
{70} and {30}.  Maximum of these minimums is 70

Second element in output indicates maximum of minimums of all
windows of size 2.
Minimums of windows of size 2 are {10}, {20}, {30}, {10}, {10},
and {30}.  Maximum of these minimums is 30

Third element in output indicates maximum of minimums of all
windows of size 3.
Minimums of windows of size 3 are {10}, {20}, {10}, {10} and {10}.
Maximum of these minimums is 20

Similarly other elements of output are computed.
```

We strongly recommend that you click here and practice it, before moving on to the solution.

A **Simple Solution** is to go through all windows of every size, find maximum of all windows. Below is C++ implementation of this idea.

```
// A naive method to find maximum of minimum of all windows of
// different sizes
#include <iostream>
using namespace std;

void printMaxOfMin(int arr[], int n)
{
    // Consider all windows of different sizes starting
    // from size 1
    for (int k=1; k<=n; k++)
    {
        // Initialize max of min for current window size k
        int maxOfMin = arr[0];

        // Traverse through all windows of current size k
        for (int i = 0; i <= n-k; i++)
        {
            // Find minimum of current window
            int min = arr[i];
            for (int j = 1; j < k; j++)
            {
                if (arr[i+j] < min)
                    min = arr[i+j];
            }

            // Update maxOfMin if required
            if (min > maxOfMin)
              maxOfMin = min;
        }

        // Print max of min for current window size
        cout << maxOfMin << " ";
    }
}

// Driver program
int main()
{
    int arr[] = {10, 20, 30, 50, 10, 70, 30};
    int n = sizeof(arr)/sizeof(arr[0]);
    printMaxOfMin(arr, n);
    return 0;
}
```

Output:

```
70 30 20 10 10 10 10
```

Time complexity of above solution can be upper bounded by O(n^3).

We can solve this problem in O(n) time using an **Efficient Solution**. The idea is to extra space. Below are detailed steps.
Step 1: Find indexes of next smaller and previous smaller for every element. Next smaller is the largest element on right side of arr[i] such that the element is smaller than arr[i]. Similarly, previous smaller element is on left side/
If there is no smaller element on right side, then next smaller is n. If there is no smaller on left side, then previous smaller is -1.

For input {10, 20, 30, 50, 10, 70, 30}, array of indexes of next smaller is {7, 4, 4, 4, 7, 6, 7}.
For input {10, 20, 30, 50, 10, 70, 30}, array of indexes of previous smaller is {-1, 0, 1, 2, -1, 4, 4}

This step can be done in O(n) time using the approach discussed in next greater element.

Step 2: Once we have indexes of next and previous smaller, we know that arr[i] is a minimum of a window of length "right[i] – left[i] – 1". Lengths of windows for which the elements are minimum are {7, 3, 2, 1, 7, 1, 2}. This array indicates, first element is minimum in window of size 7, second element is minimum in window of size 1, and so on.

Create an auxiliary array ans[n+1] to store the result. Values in ans[] can be filled by iterating through right[] and left[]

```
    for (int i=0; i < n; i++)
    {
        // length of the interval
        int len = right[i] - left[i] - 1;

        // a[i] is the possible answer for
        // this length len interval
        ans[len] = max(ans[len], arr[i]);
    }
```

We get the ans[] array as {0, 70, 30, 20, 0, 0, 0, 10}. Note that ans[0] or answer for length 0 is useless.

Step 3: Some entries in ans[] are 0 and yet to be filled. For example, we know maximum of minimum for lengths 1, 2, 3 and 7 are 70, 30, 20 and 10 respectively, but we don't know the same for lengths 4, 5 and 6.
Below are few important observations to fill remaining entries
a) Result for length i, i.e. ans[i] would always be greater or same as result for length i+1, i.e., an[i+1].
b) If ans[i] is not filled it means there is no direct element which is minimum of length i and therefore either the element of length ans[i+1], or ans[i+2], and so on is same as ans[i]
So we fill rest of the entries using below loop.

```
    for (int i=n-1; i>=1; i--)
        ans[i] = max(ans[i], ans[i+1]);
```

Below is C++ implementation of above algorithm.

```
// An efficient C++ program to find maximum of all minimums of
// windows of different sizes
#include <iostream>
#include<stack>
using namespace std;

void printMaxOfMin(int arr[], int n)
{
    stack<int> s; // Used to find previous and next smaller

    // Arrays to store previous and next smaller
    int left[n+1];
    int right[n+1];

    // Initialize elements of left[] and right[]
    for (int i=0; i<n; i++)
    {
        left[i] = -1;
        right[i] = n;
    }

    // Fill elements of left[] using logic discussed on
    // http://www.geeksforgeeks.org/next-greater-element/
    for (int i=0; i<n; i++)
    {
        while (!s.empty() && arr[s.top()] >= arr[i])
            s.pop();

        if (!s.empty())
            left[i] = s.top();

        s.push(i);
    }

    // Empty the stack as stack is going to be used for right[]
    while (!s.empty())
        s.pop();

    // Fill elements of right[] using same logic
    for (int i = n-1 ; i>=0 ; i-- )
    {
        while (!s.empty() && arr[s.top()] >= arr[i])
            s.pop();

        if(!s.empty())
```

```
            right[i] = s.top();

        s.push(i);
    }

    // Create and initialize answer array
    int ans[n+1];
    for (int i=0; i<=n; i++)
        ans[i] = 0;

    // Fill answer array by comparing minimums of all
    // lengths computed using left[] and right[]
    for (int i=0; i<n; i++)
    {
        // length of the interval
        int len = right[i] - left[i] - 1;

        // arr[i] is a possible answer for this length
        // 'len' interval, check if arr[i] is more than
        // max for 'len'
        ans[len] = max(ans[len], arr[i]);
    }

    // Some entries in ans[] may not be filled yet. Fill
    // them by taking values from right side of ans[]
    for (int i=n-1; i>=1; i--)
        ans[i] = max(ans[i], ans[i+1]);

    // Print the result
    for (int i=1; i<=n; i++)
        cout << ans[i] << " ";
}

// Driver program
int main()
{
    int arr[] = {10, 20, 30, 50, 10, 70, 30};
    int n = sizeof(arr)/sizeof(arr[0]);
    printMaxOfMin(arr, n);
    return 0;
}
```

Output:

```
70 30 20 10 10 10 10
```

Time Complexity: O(n)
Auxiliary Space: O(n)