Efficient Solution: We can solve this problem in O(n) time. The idea is to use extra space. Below are detailed steps.
Step 1: Find indexes of next smaller and previous smaller for every element. Next smaller is the nearest smallest element on right side of arr[i]. Similarly, a previous smaller element is the nearest smallest element on the left side of arr[i].
If there is no smaller element on the right side, then the next smaller is n. If there is no smaller on the left side, then the 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, the first element is minimum in the window of size 7, the second element is minimum in the window of size 3, 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;

    // arr[i] is a 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., ans[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 implementation of above algorithm.