Q3 Largest rectangular area in histogram

Before solving this question, we need to solve next smaller element in the array.

2	1	4	3

Ans → { 1, -1, 3, -1 }

Brute force approach can be using 2 nested for loops and hence $O(n^2)$ time complexity but we need to do in $O(n)$ time.

Left to right ⇒ $O(n^2)$
Right to left ⇒ ?

Using stack
1) Push -1 initially in the stack.
2) For each element in array we get, search in stack for the smaller element. Once we get smaller element simply push that element.

(I)

-1

3 > -1, yes store it and push 3 in stack.

(II)

3
-1

4 > 3. Yes store it & push 4 in the stack

(III)

4
3
-1

1 > 4. No pop it.

3
-1

1 > 3. No pop it.

-1

1 > -1. Yes store the answer and hence push the stack.

(IV)

1
-1

2 > 1. Yes store the answer and hence push in the stack.

Now we have traversed all the array elements and ans is
{1, -1, 3, -1}

Code

```
vector <int> v {2, 1, 4, 3};
stack <int> s;
vector <int> ans (v.size());
//Initialey push -1 in the stack
s.push(-1);
//Traverse from right to left
for (int i = v.size()-1; i>=0; i--){
    //Processing element in vector
    int curr = v[i];
    //See step 3 and 4 in dry run
    while (s.top() >= curr) {
```

```
        s.pop();
    }
    //Store Top element in ans vector.
    ans[i] = s.top(); //Smaller element found
    s.push(curr);
}
//Print ans vector
for (int i=0 ; i< ans.size() ; i++){
        cout << ans[i] << " ";
}
```

Prev smaller element

2	1	4	3

ans → {-1, -1, 1, 1}

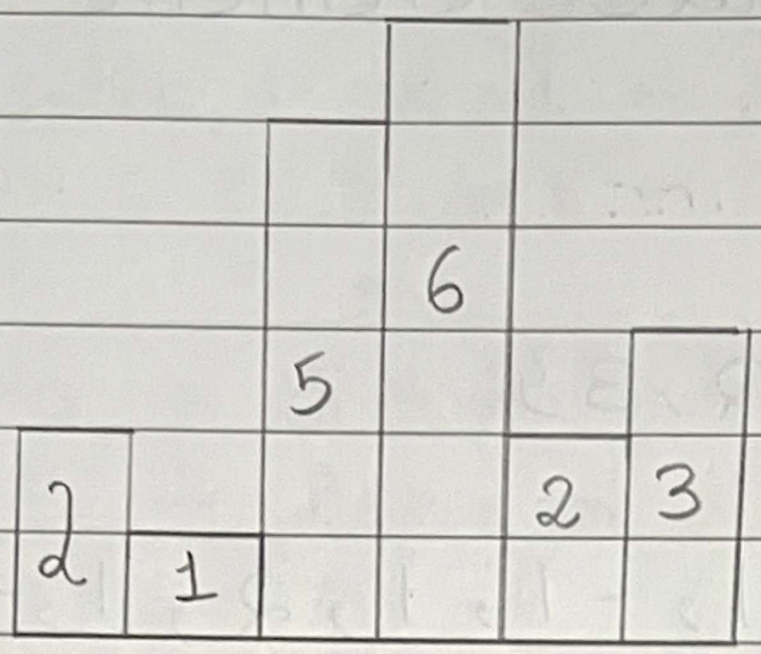
We just have to change the direction of traversal as we have done in the next smaller element. The code inside the for loop remains same.

```
for (int i=0 ; i< v.size() ; i++){
        //Same code → Next smaller element
}
```

TC = O(n) → For a single element only 1 time push and pop is done, hence 2n operation for n elements. O(2n) → O(n)

V.V.V.V Imp Q → Largest area in histogram
↳ Rectangular

i/p → {2, 1, 5, 6, 2, 3}
O/p → 10



This histogram is drawn from the i/p array. Each element in i/p array indicates the height of histogram

Brute force

A1 = Consider the 1st histogram of height 2. We can't extend the width on left & right both. Hence area = 2 × 1 = 2

A2 = Consider the 2nd histogram of height 1. We can extend on the left by 1 unit and on right by 4 unit. Hence Area = 1 × 6 = 6

A3 = Consider the 3rd histogram of height 5. Can't extend on left but can be extended on right by 1 unit. Hence area = 5 × 2 = 10.

A4 = Can't extend on both sides & hence area = 1 × 6 = 6

A5 = Can be extended on left by 2 units & on right by 1 unit. Hence area = 2 × 4 = 8

A6 = Can't be extended on both sides & hence area = 1 × 3 = 3

Now the condition for extending on the left is until we find the prev smaller element & to extend in the right is to

find the next smaller element.

Optimal solution

i/p → {2, 1, 5, 6, 2, 3}

prev smaller element index → {-1, -1, 1, 2, 1, 4}

next smaller element index → {1, -1, 4, 4, -1, -1}

In the case of next smaller element instead of the index -1 i.e invalid index, we will be storing the size.

next smaller element index {1, 6, 4, 4, 6, 6}

* Now simply traverse the array.

✓ i = 0 → element = 2 which is the length
width = next - prev - 1
width = 1 - (-1) - 1
width = 2 - 1 = 1

Area = 2 × 1 = 2

✓ i = 1 → length = 1
width = next - prev - 1
= 6 - (-1) - 1
= 7 - 1
= 6
Area = 1 × 6 = 6

✓ $i = 2 \rightarrow$ length $= 5$
width = next − prev − 1
= 4 − 1 − 1
= 2
Area = 5 × 2 = 10

✓ $i = 3 \rightarrow$ length = 6
width = next − prev − 1
= 4 − 2 − 1
= 1
Area = 6 × 1 = 6

✓ $i = 4 \rightarrow$ length = 2
width = next − prev − 1
= 6 − 1 − 1
= 4
Area = 2 × 4 = 8

✓ $i = 5 \rightarrow$ length = 3
width = next − prev − 1
= 6 − 4 − 1
= 1
Area = 3 × 1 = 3

Maximum area found is 10.

Code

```
vector <int> prevSmallerElement (vector<int>
                                  &v){
    stack <int> s;
    s.push (-1);
    vector <int> ans (v.size());
    for (int i=0 ; i<v.size() -1 ; i++){
        int curr = v[i];
```

```
//Changes done below
while (s.top() != -1 && v[s.top()]
        ↳ Valid index          >= curr){
        s.pop();
}
ans[i] = s.top();
//Push the index
s.push(i);
}
return ans;
}

vector <int> nextSmallerElement (vector <int>&
{
 stack <int> s;
 s.push(-1);
 vector <int> ans(v.size());
 for (int i = v.size()-1; i >= 0; i--){
      int curr = v[i];
      //Changes done below
      while (s.top() != -1 && v[s.top()] >=
                                   curr){
           s.pop();
      }
      ans[i] = s.top();
      //Push the index
      s.push(i);
 }
 return ans;
}

int largestArea (vector <int> & heights){
```

```
// Previous smaller element array
vector <int> prev = prevSmallerElement(heights);
// Next smaller element array
vector <int> next = nextSmallerElement (heights);
// Finding maximum area
int size = heights.size();
int maxArea = INT_MIN;
for (int i=0; i< heights.size(); i++){
    int length = heights [i];
    if (next [i] == -1){
        next [i] = size;
    }
    int width = next [i] - prev [i] - 1;
    int area = length * width;
    maxArea = max (maxArea, area);
}
return maxArea;
}
```

Updation step