Area of largest rectangle in Histogram

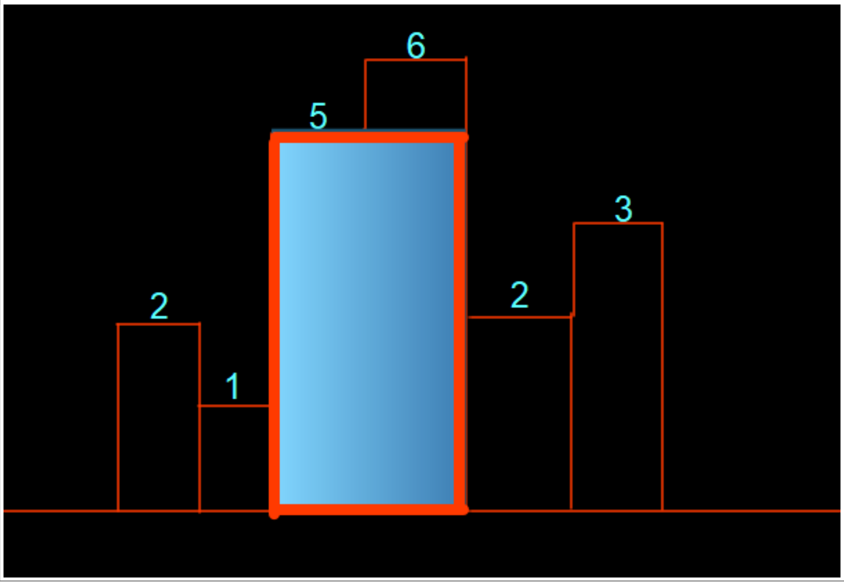
Problem Statement: Given an array of integers heights representing the histogram’s bar height where the width of each bar is 1 return the area of the largest rectangle in histogram.

Example:

Input: N =6, heights[] = {2,1,5,6,2,3}

Output: 10

Explanation:

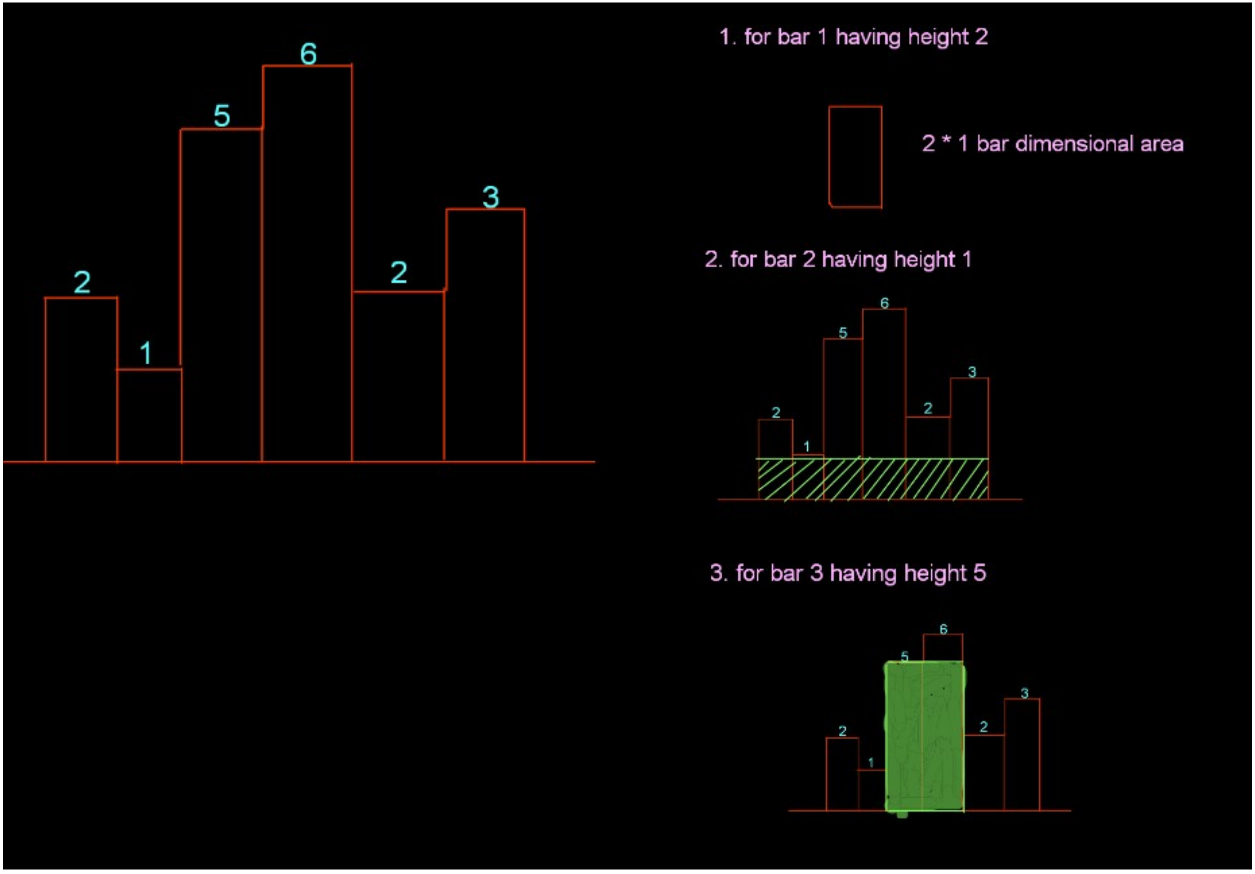


Solution

***Disclaimer**: Don’t jump directly to the solution, try it out yourself first.*

Solution 1: Brute Force Approach

Intuition: The intuition behind the approach is taking different bars and finding the maximum width possible using the bar.

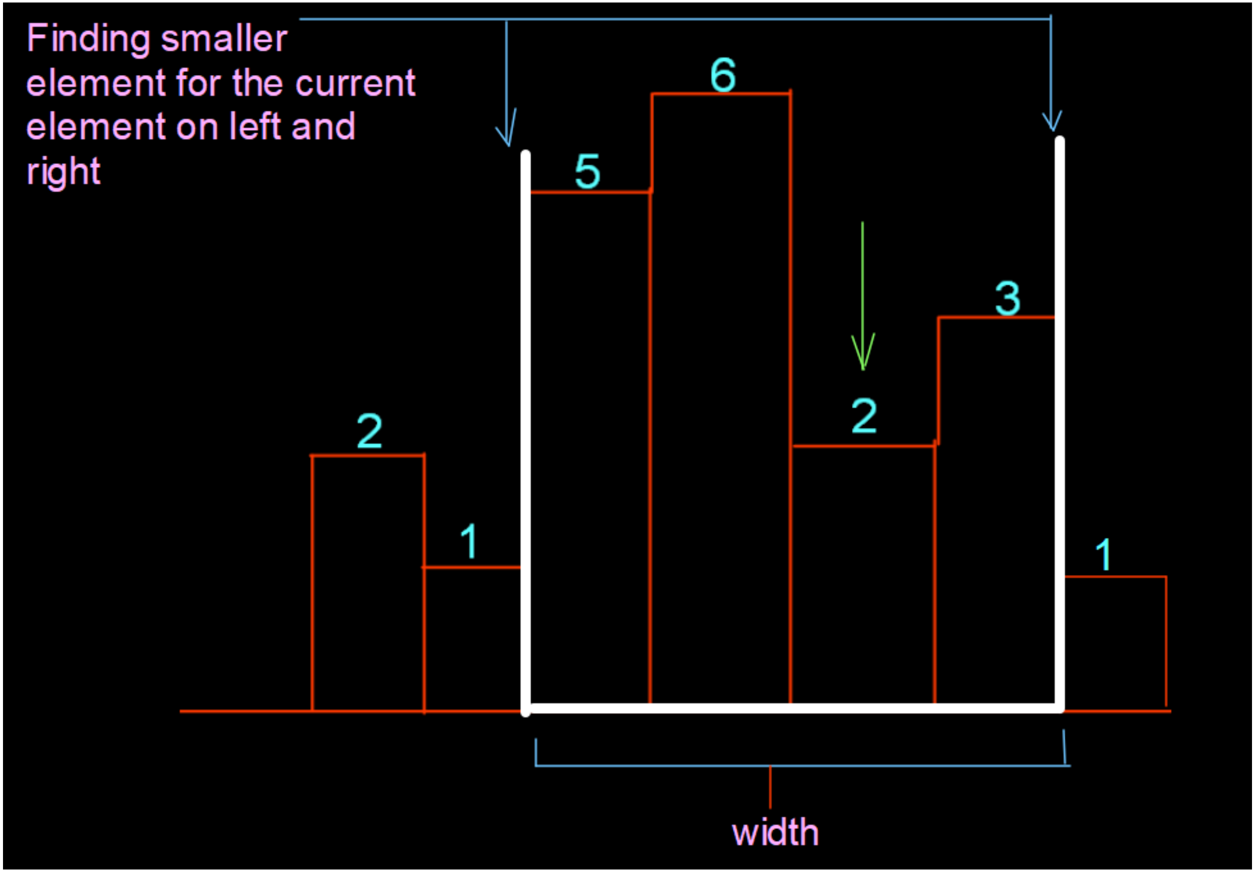

6
5
3
2
2
1
1. for bar 1 having height 2
2 * 1 bar dimensional area
2. for bar 2 having height 1
3. for bar 3 having height 5

Similarly for other bars, we will find the areas possible:-
Considering the width of each bar as 1 unit.

For first bar, area possible = 2* 1 =2 sq . units
For second bar, area possible = 1 * 6 =6 sq . units
For third bar , area possible = 5 *2 = 10 sq . units
For fourth bar , area possible = 6 * 1 = 6 sq . units
For Fifth bar , area possible = 2 * 4 = 8 sq . units
For Sixth bar , area possible = 3 * 1 =3 sq . units
So, the maximum area possible = 10 sq units.

Approach:

The approach is to find the right smaller and left smaller element and find the largest Rectangle area in Histogram.


Finding smaller element for the current element on left and right
2
1
5
6
2
3
1
width

Code:

- C++ Code
- Java Code

```
#include <bits/stdc++.h>

using namespace std;
// Brute Force Approach to find largest rectangle area in Histogram
int largestarea(int arr[], int n) {
   int maxArea = 0;
   for (int i = 0; i < n; i++) {
      int minHeight = INT_MAX;
      for (int j = i; j < n; j++) {
         minHeight = min(minHeight, arr[j]);
         maxArea = max(maxArea, minHeight * (j - i + 1));
      }
   }
   return maxArea;
}
int main() {
   int arr[] = {2, 1, 5, 6, 2, 3, 1};
   int n = 7;
   cout << "The largest area in the histogram is " << largestarea(arr, n); // Printing the largest rectangle area
   return 0;
}
```

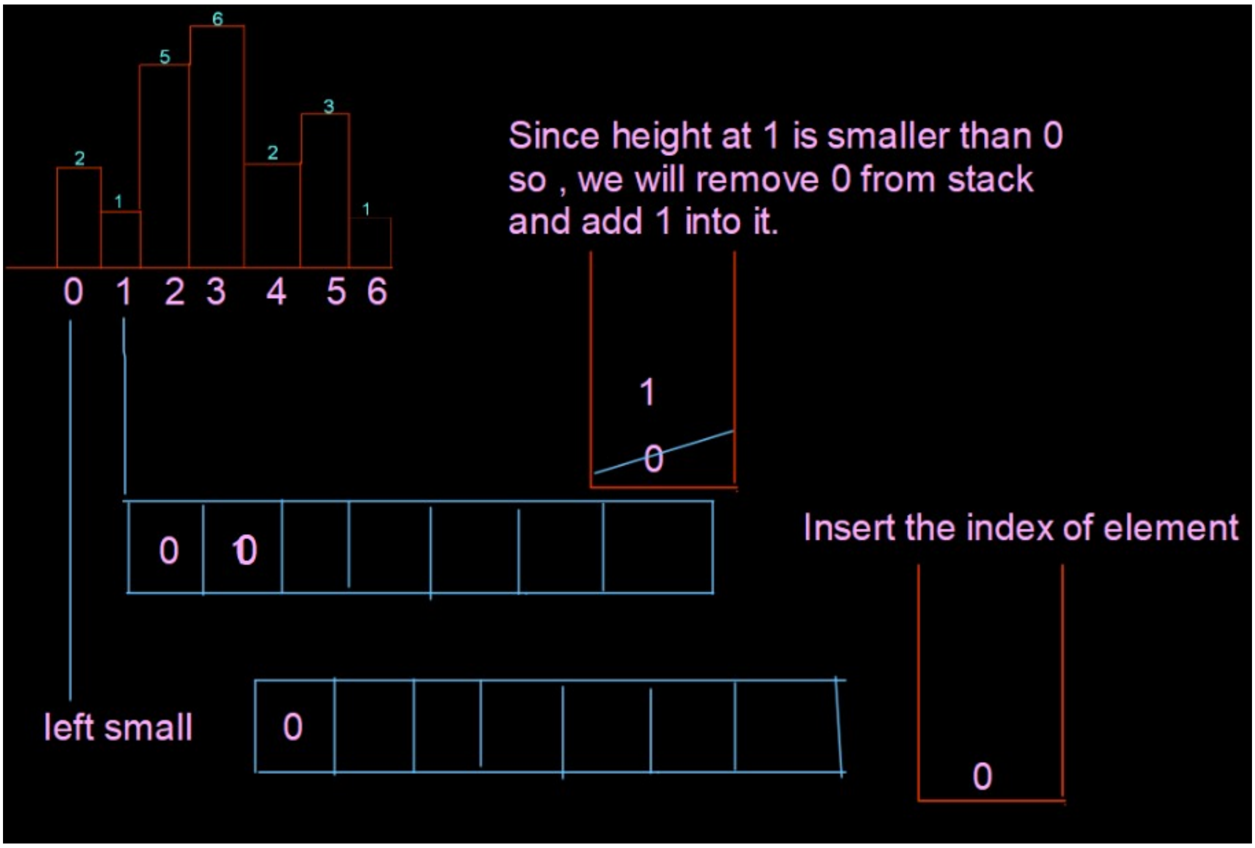
Output: The largest area in the histogram is 10

Time Complexity: O(N*N)

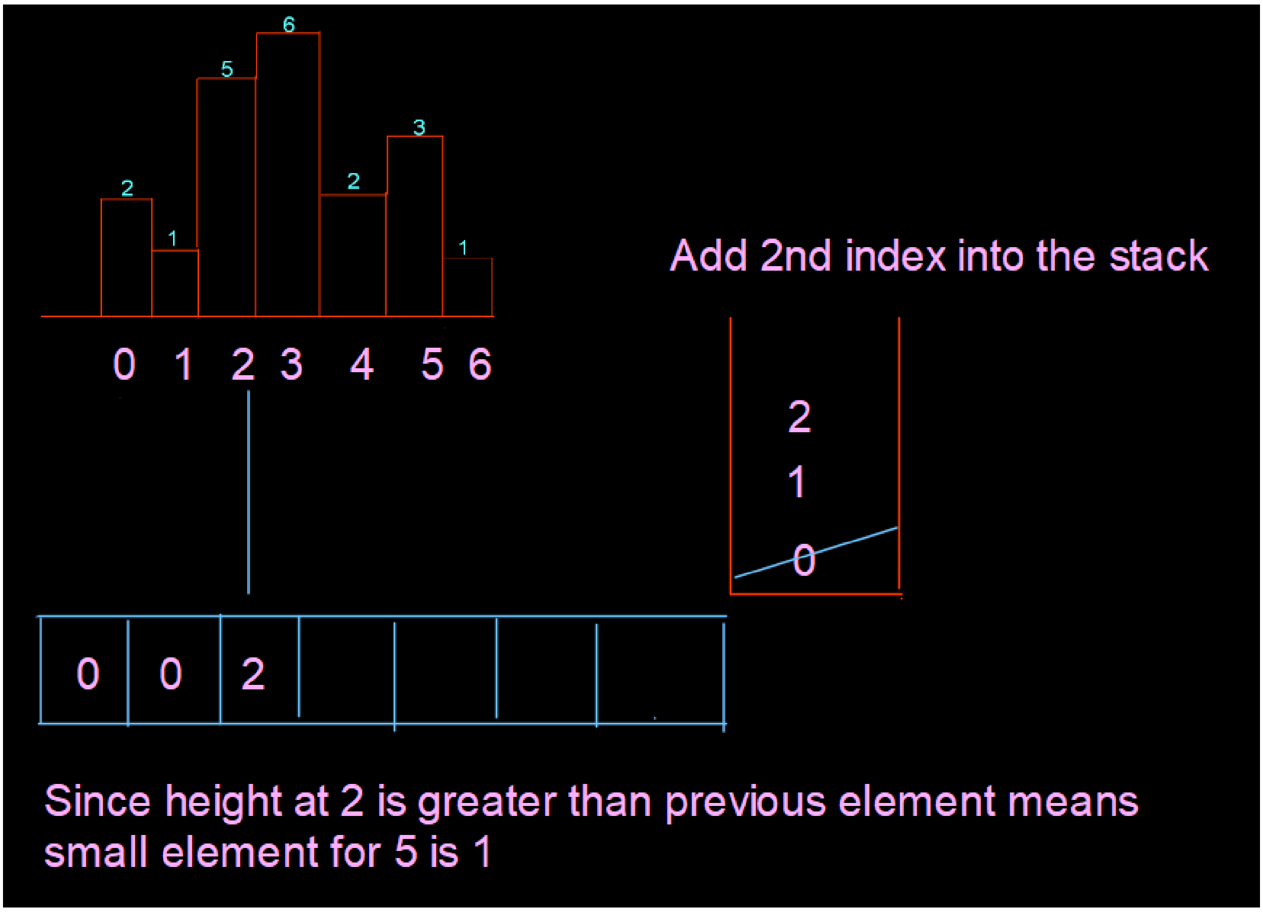
Space Complexity: O(1)

Solution 2: Optimised Approach 1
Intuition: The intuition behind the approach is the same as finding the smaller element on both sides but in an optimized way using the concept of the next greater element and the next smaller element.
Approach:

1.**Steps to be done for finding Left smaller element**

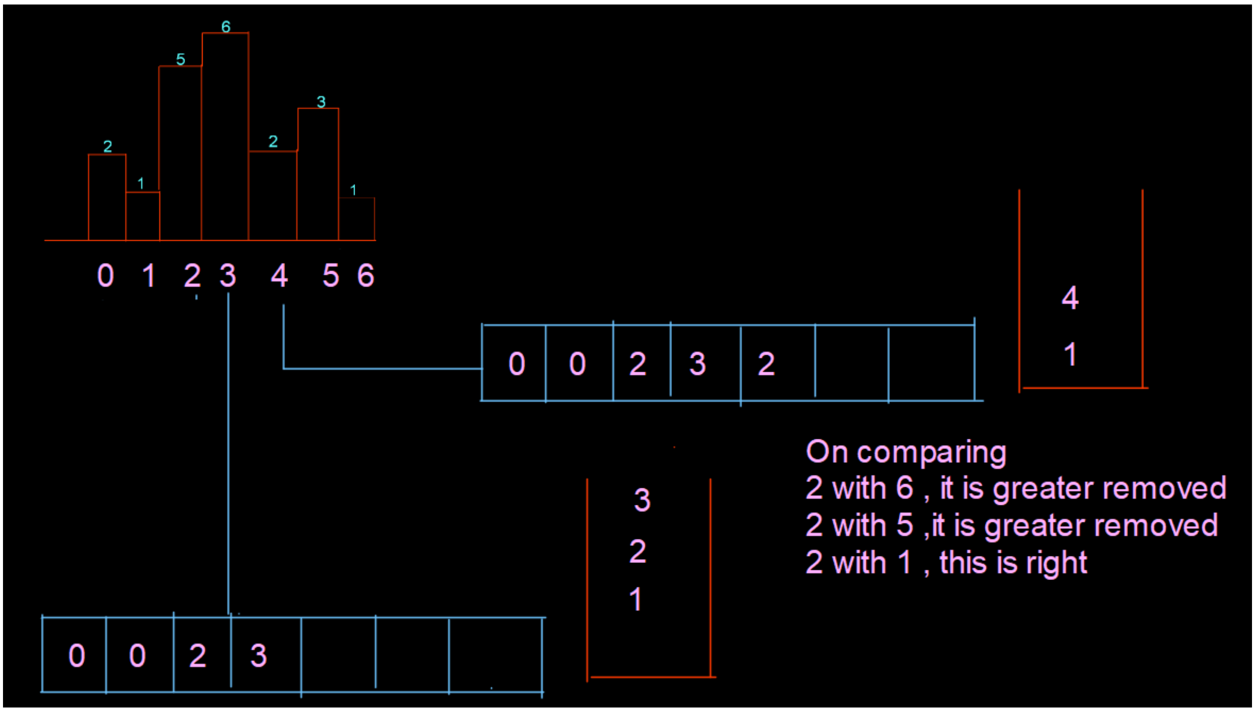


6
5
3
2
2
1
1
0 1 2 3 4 5 6
Since height at 1 is smaller than 0 so , we will remove 0 from stack and add 1 into it.
1
0
0
10
Insert the index of element
left small
0
0



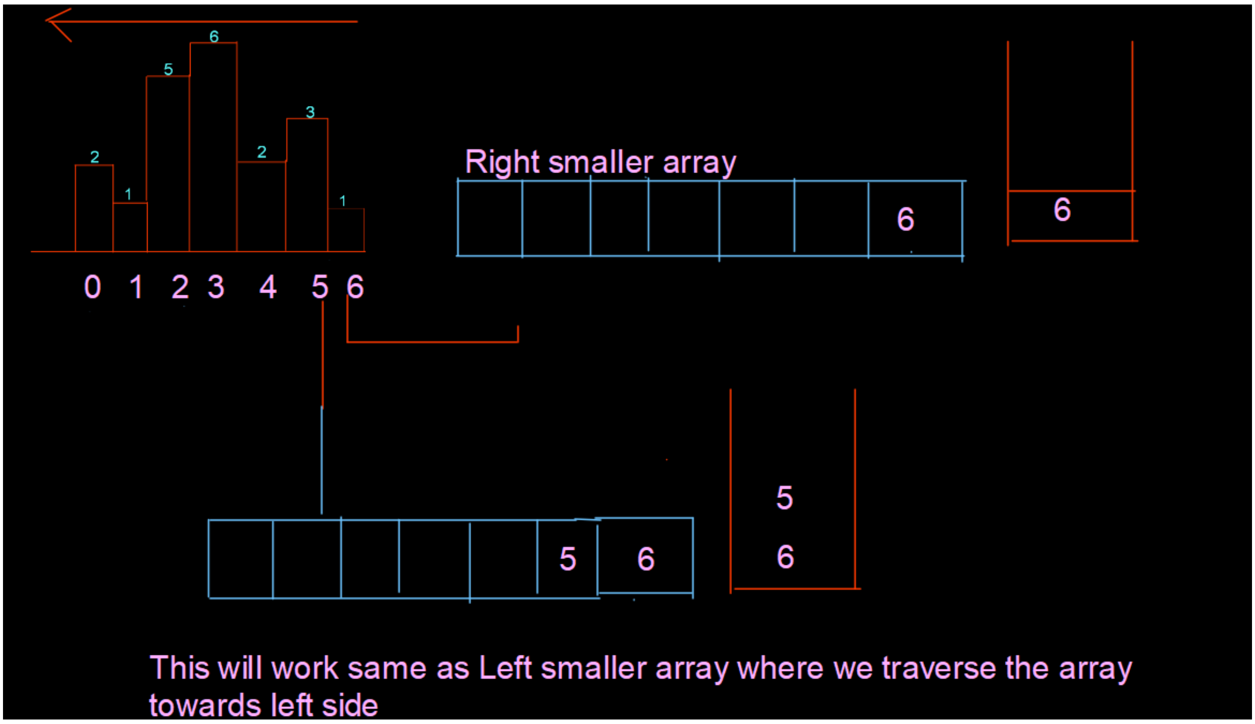
Add 2nd index into the stack

Since height at 2 is greater than previous element means small element for 5 is 1



On comparing
2 with 6 , it is greater removed
2 with 5 ,it is greater removed
2 with 1 , this is right

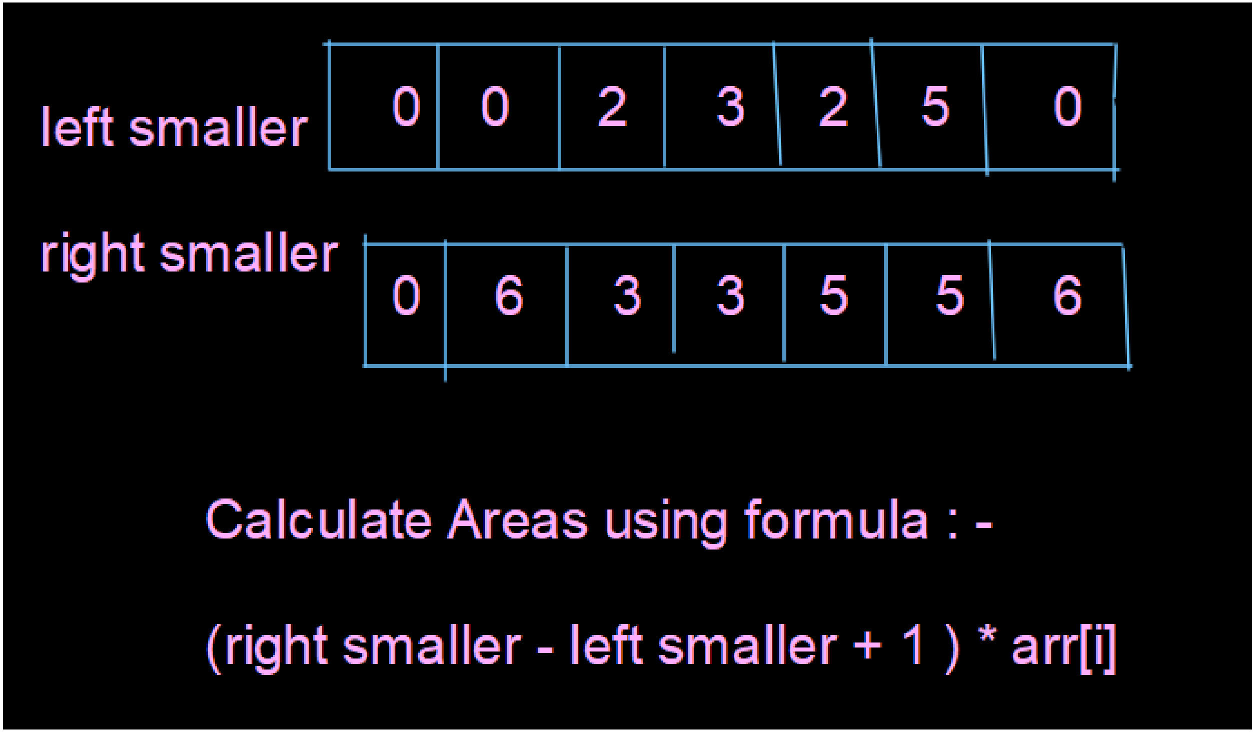
2. **Steps to be done for finding the Right smaller element**



Right smaller array

This will work same as Left smaller array where we traverse the array towards left side

After finding the right smaller and left smaller of each subsequent array elements, we



left smaller	0	0	2	3	2	5	0
right smaller	0	6	3	3	5	5	6

Calculate Areas using formula : -

(right smaller - left smaller + 1) * arr[i]

Area for first index – (0 – 0 +1) * 2 = 2
Area for second index – (6 – 0 + 1) * 1 = 6
Area for third index – (3 – 2 +1) * 5 = 10
Area for fourth index – (3 – 3 + 1) * 6 = 6
Area for fifth index – (5 – 2 +1) * 2 = 8
Area for sixth index – (5 – 5 + 1) * 3 = 3
Area for seventh index – (6 – 0 +1) * 1 = 7
So, the maximum area out of these is 10 sq units.

Code:

- C++ Code
- Java Code

```
#include <bits/stdc++.h>

using namespace std;
class Solution {
  public:
    int largestRectangleArea(vector < int > & heights) {
      int n = heights.size();
      stack < int > st;
      int leftsmall[n], rightsmall[n];
      for (int i = 0; i < n; i++) {
        while (!st.empty() && heights[st.top()] >= heights[i]) {
          st.pop();
        }
        if (st.empty())
          leftsmall[i] = 0;
        else
          leftsmall[i] = st.top() + 1;
        st.push(i);
      }
      // clear the stack to be re-used
      while (!st.empty())
        st.pop();

      for (int i = n - 1; i >= 0; i--) {
        while (!st.empty() && heights[st.top()] >= heights[i])
          st.pop();

        if (st.empty())
          rightsmall[i] = n - 1;
        else
          rightsmall[i] = st.top() - 1;

        st.push(i);
      }
      int maxA = 0;
      for (int i = 0; i < n; i++) {
        maxA = max(maxA, heights[i] * (rightsmall[i] - leftsmall[i] + 1));
      }
      return maxA;
    }
};
int main() {
  vector<int> heights = {2, 1, 5, 6, 2, 3, 1};
  Solution obj;
  cout << "The largest area in the histogram is " << obj.largestRectangleArea(heights);
  return 0;
}
```

Output: The largest area in the histogram is 10

Time Complexity: O(N)

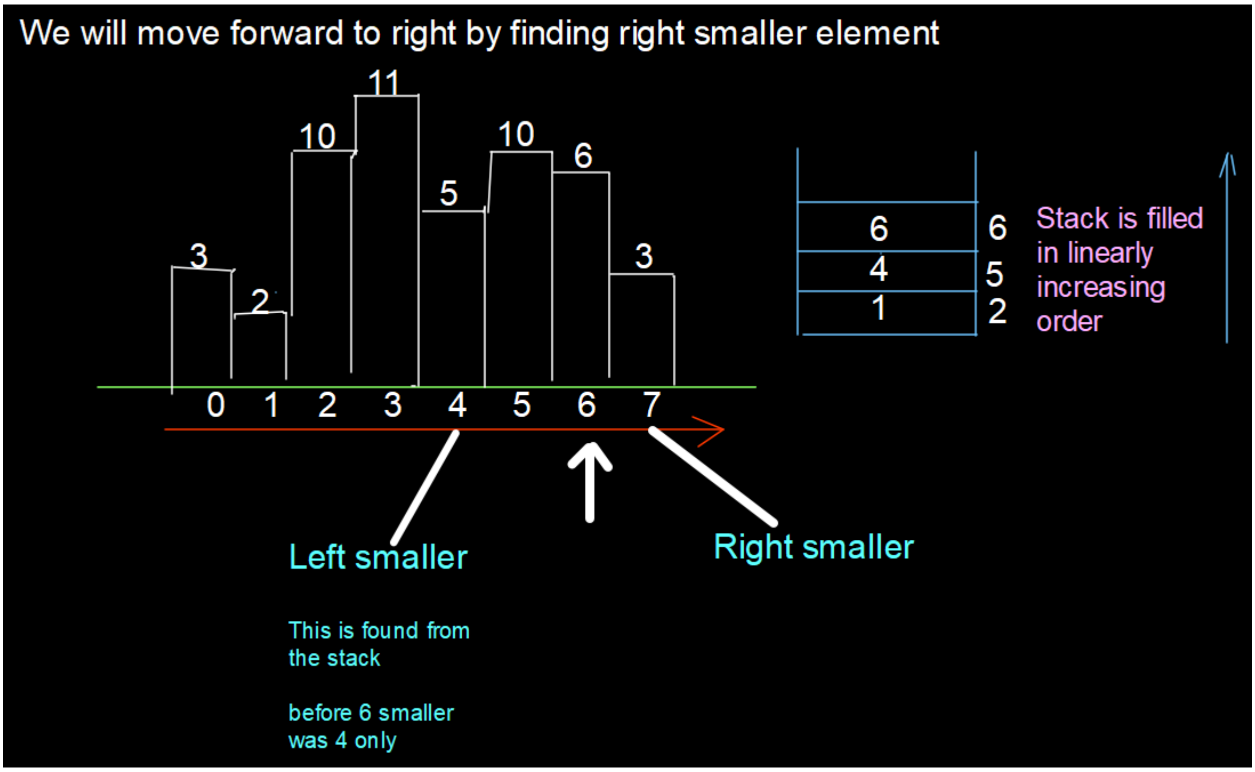
Space Complexity: O(3N) where 3 is for the stack, left small array and a right small array

Solution 3: Optimised approach 2

Intuition:

This approach is a single pass approach instead of a two-pass approach. When we traverse the array by finding the next greater element, we found that some elements were inserted into the stack which signifies that after them the smallest element is themselves

So we can find the area of the rectangle by using **arr[i] * (right smaller – left smaller -1)**.



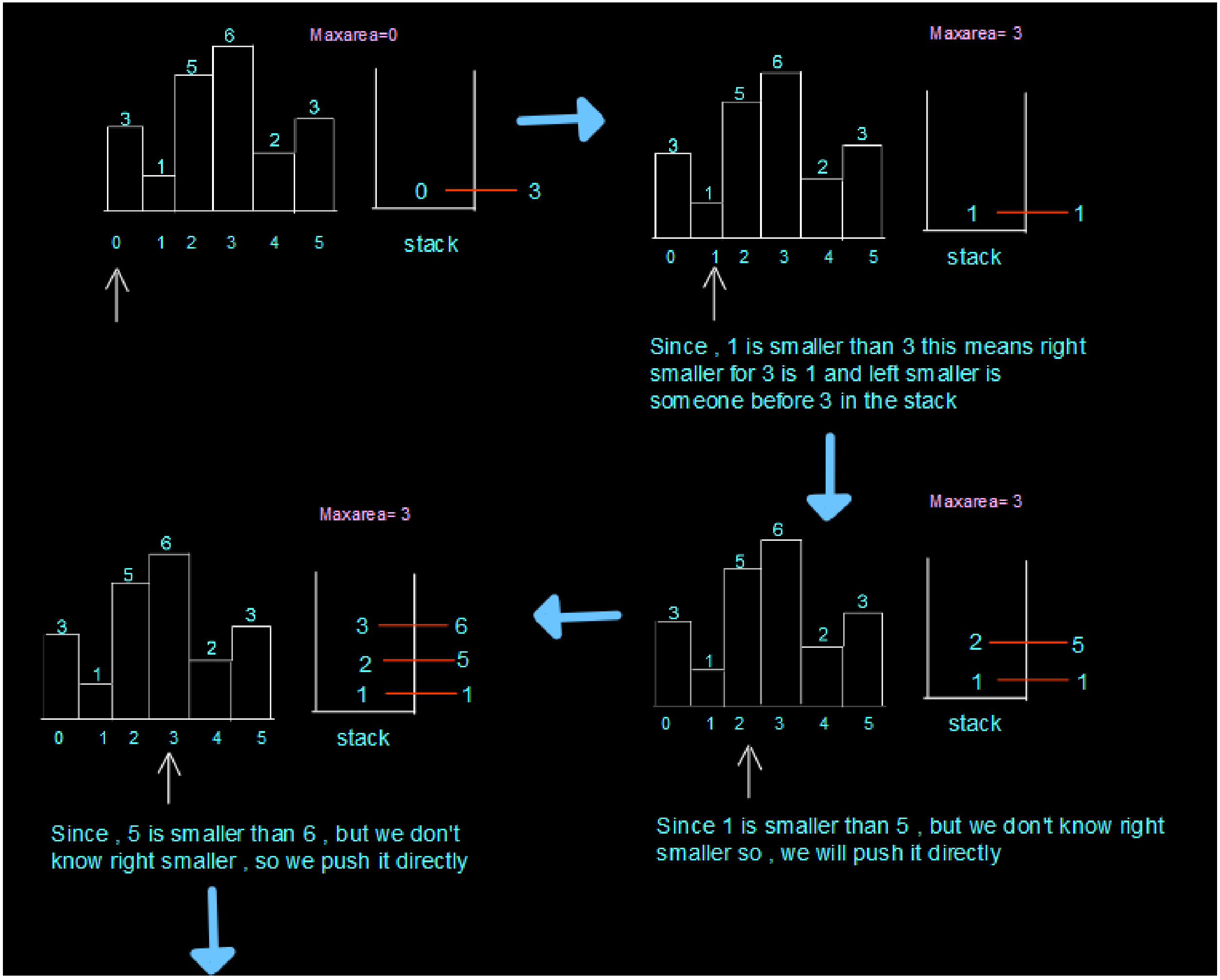
We will move forward to right by finding right smaller element

Stack is filled in linearly increasing order

This is found from the stack

before 6 smaller was 4 only

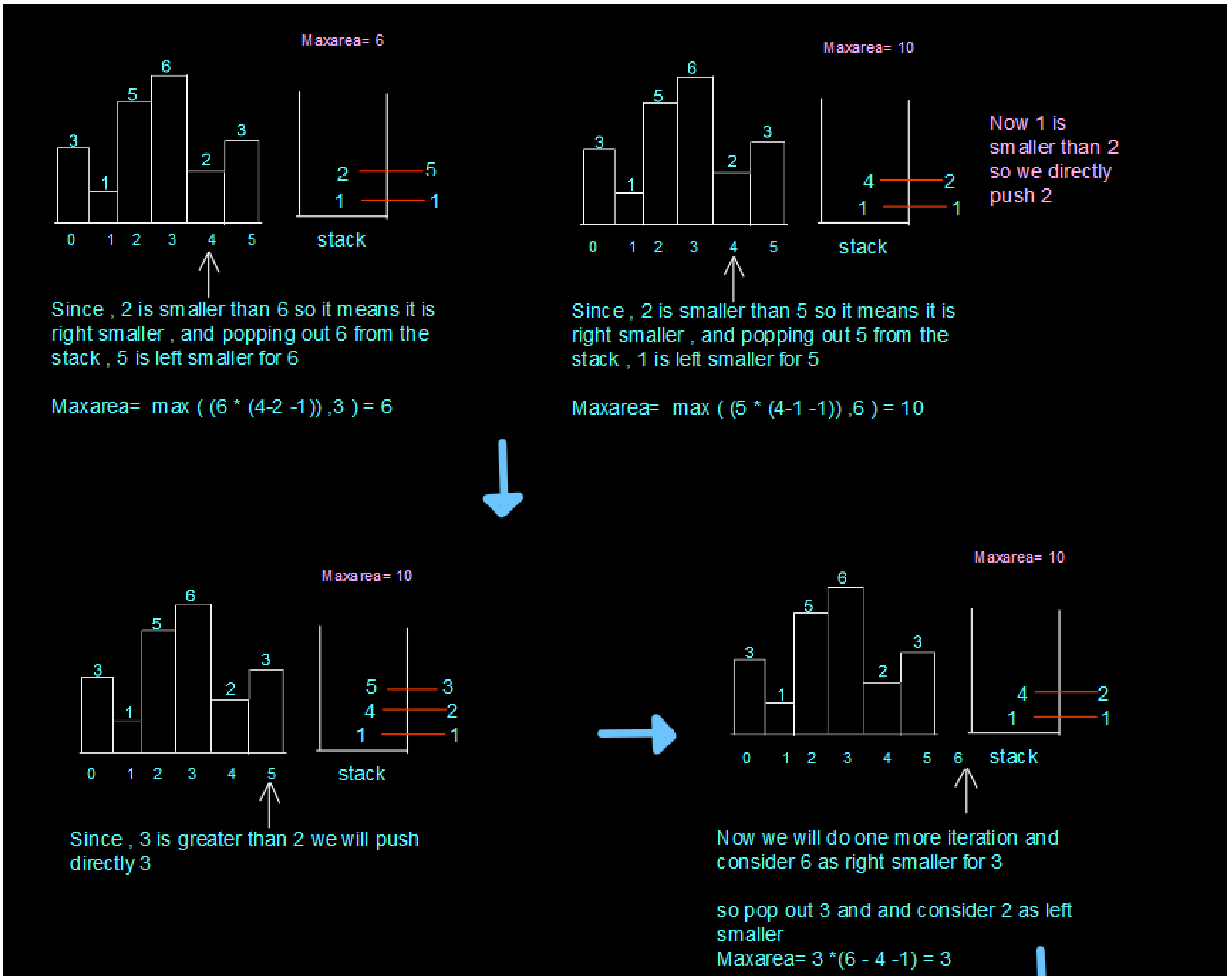
Approach:



Since , 1 is smaller than 3 this means right smaller for 3 is 1 and left smaller is someone before 3 in the stack

Since 1 is smaller than 5 , but we don't know right smaller so , we will push it directly

Since , 5 is smaller than 6 , but we don't know right smaller , so we push it directly



Since , 2 is smaller than 6 so it means it is right smaller , and popping out 6 from the stack , 5 is left smaller for 6

Maxarea= max ((6 * (4-2 -1)) ,3) = 6

Since , 2 is smaller than 5 so it means it is right smaller , and popping out 5 from the stack , 1 is left smaller for 5

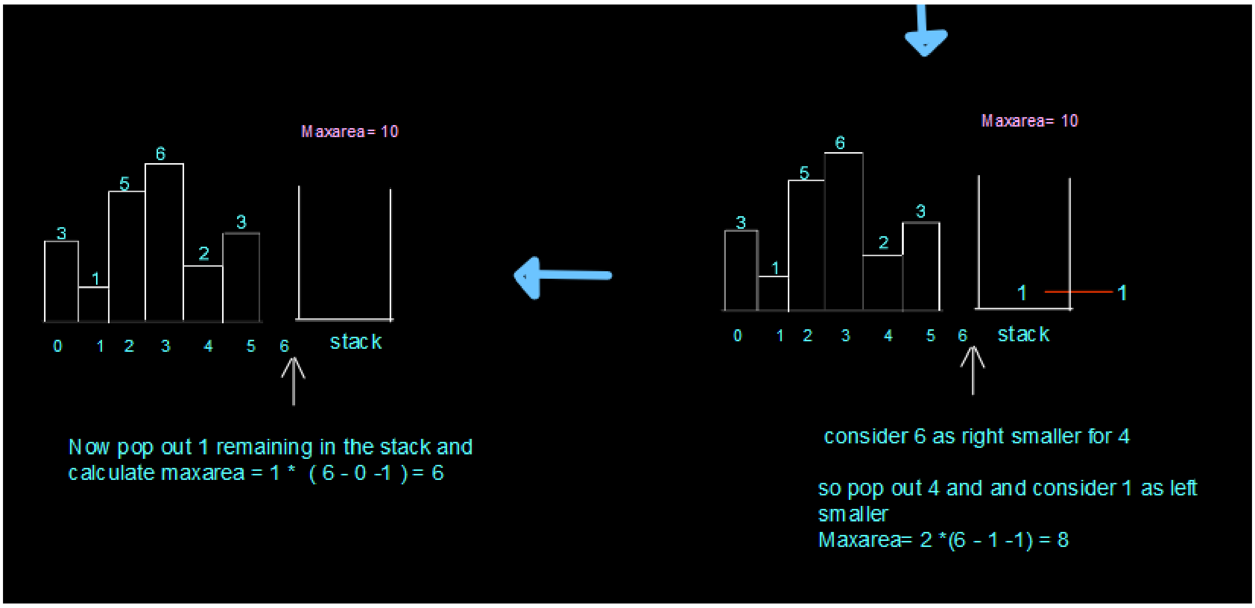
Maxarea= max ((5 * (4-1 -1)) ,6) = 10

Now 1 is smaller than 2 so we directly push 2

Since , 3 is greater than 2 we will push directly 3

Now we will do one more iteration and consider 6 as right smaller for 3

so pop out 3 and and consider 2 as left smaller
Maxarea= 3 *(6 - 4 -1) = 3



consider 6 as right smaller for 4

so pop out 4 and and consider 1 as left smaller
Maxarea= 2 *(6 - 1 -1) = 8

Now pop out 1 remaining in the stack and calculate maxarea = 1 * (6 - 0 -1) = 6

Code:

- C++ Code
- Java Code

```
#include <bits/stdc++.h>

using namespace std;
class Solution {
  public:
    int largestRectangleArea(vector < int > & histo) {
      stack < int > st;
      int maxA = 0;
      int n = histo.size();
      for (int i = 0; i <= n; i++) {
        while (!st.empty() && (i == n || histo[st.top()] >= histo[i])) {
          int height = histo[st.top()];
          st.pop();
          int width;
          if (st.empty())
            width = i;
          else
            width = i - st.top() - 1;
          maxA = max(maxA, width * height);
        }
        st.push(i);
      }
      return maxA;
    }
};
int main() {
  vector < int > histo = {2, 1, 5, 6, 2, 3, 1};
  Solution obj;
  cout << "The largest area in the histogram is " << obj.largestRectangleArea(histo) << endl;
  return 0;
}
```

Output: The largest area in the histogram is 10

Time Complexity: O(N) + O (N)

Space Complexity: O(N)