Area of largest rectangle in Histogram

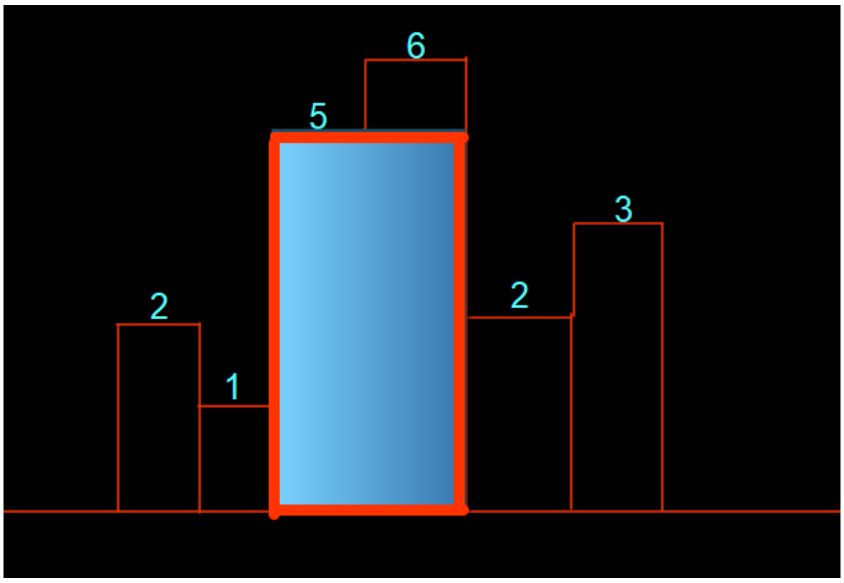
Problem Statement: Given an array of integers heights representing the histogram's bar height where the width of each bar is 1 return the area of the largest rectangle in histogram.

Example:

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

`Output: 10`

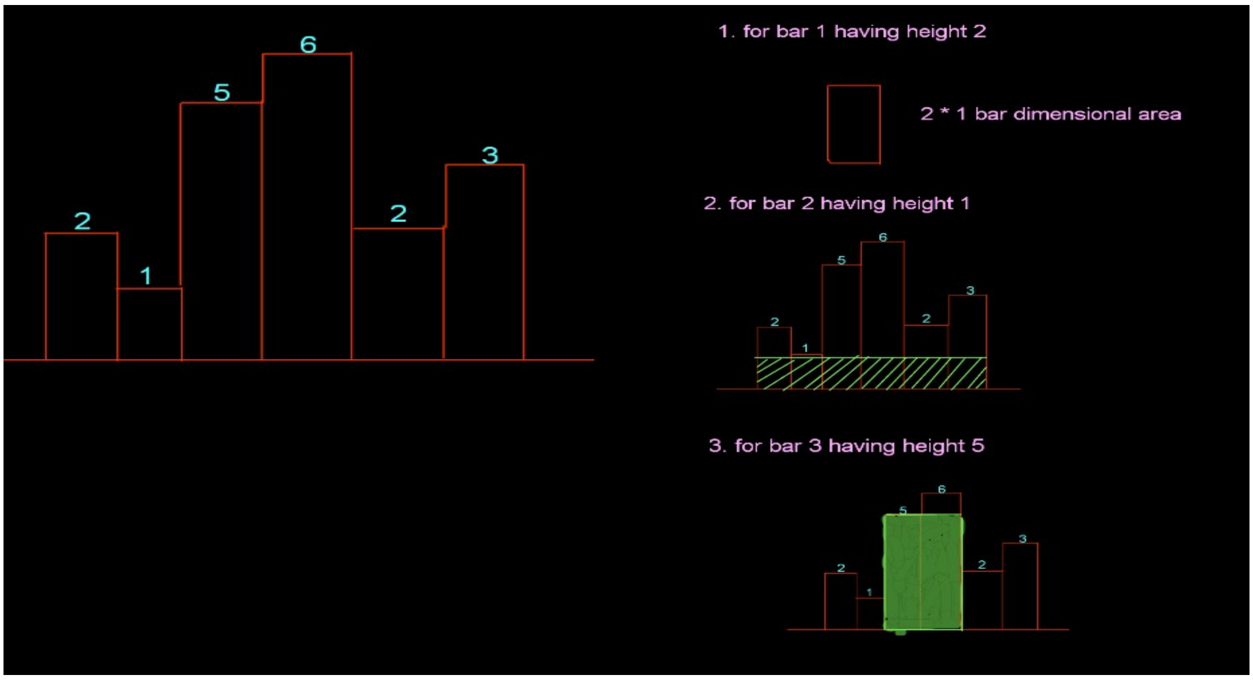
`Explanation:`



Solution

Solution 1: Brute Force Approach

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



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
6
5
3
2
1
2

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

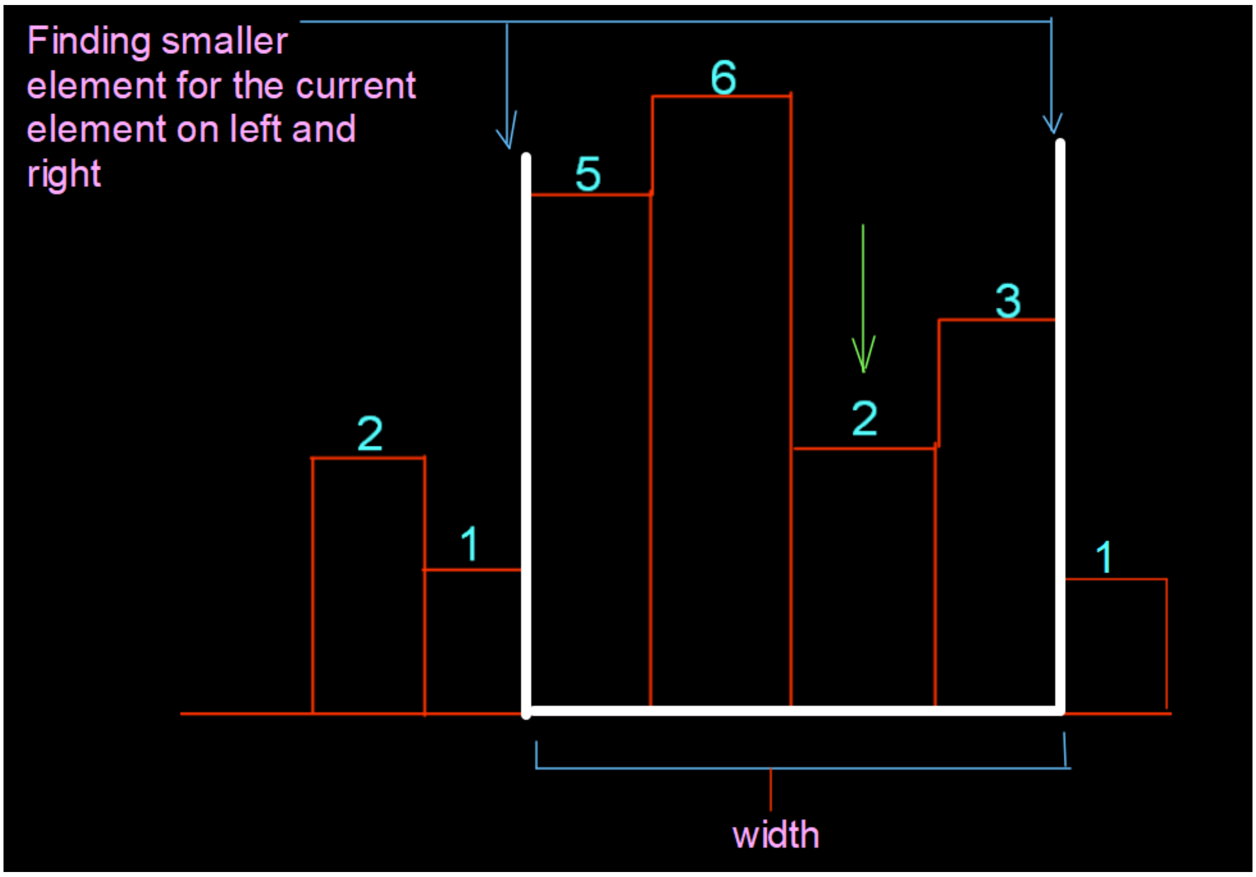
For Fifth bar , area possible = 2 * 4 = 8 sq . units

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:

```
import java.util.*;
// Brute Force Approach to find largest rectangle area in Histogram
public class Main {
    static int largestarea(int arr[], int n) {
        int maxArea = 0;
        for (int i = 0; i < n; i++) {
            int minHeight = Integer.MAX_VALUE;
            for (int j = i; j < n; j++) {
                minHeight = Math.min(minHeight, arr[j]);
                maxArea = Math.max(maxArea, minHeight * (j - i + 1));
            }
        }
        return maxArea;
    }
    public static void main(String args[]) {
        int arr[] = {2, 1, 5, 6, 2, 3, 1};
        int n = 7;
        System.out.println("The largest area in the histogram is " +
largestarea(arr, n)); // Printing the largest rectangle area

    }
}
```

Output: The largest area in the histogram is 10

Time Complexity: O(N*N)

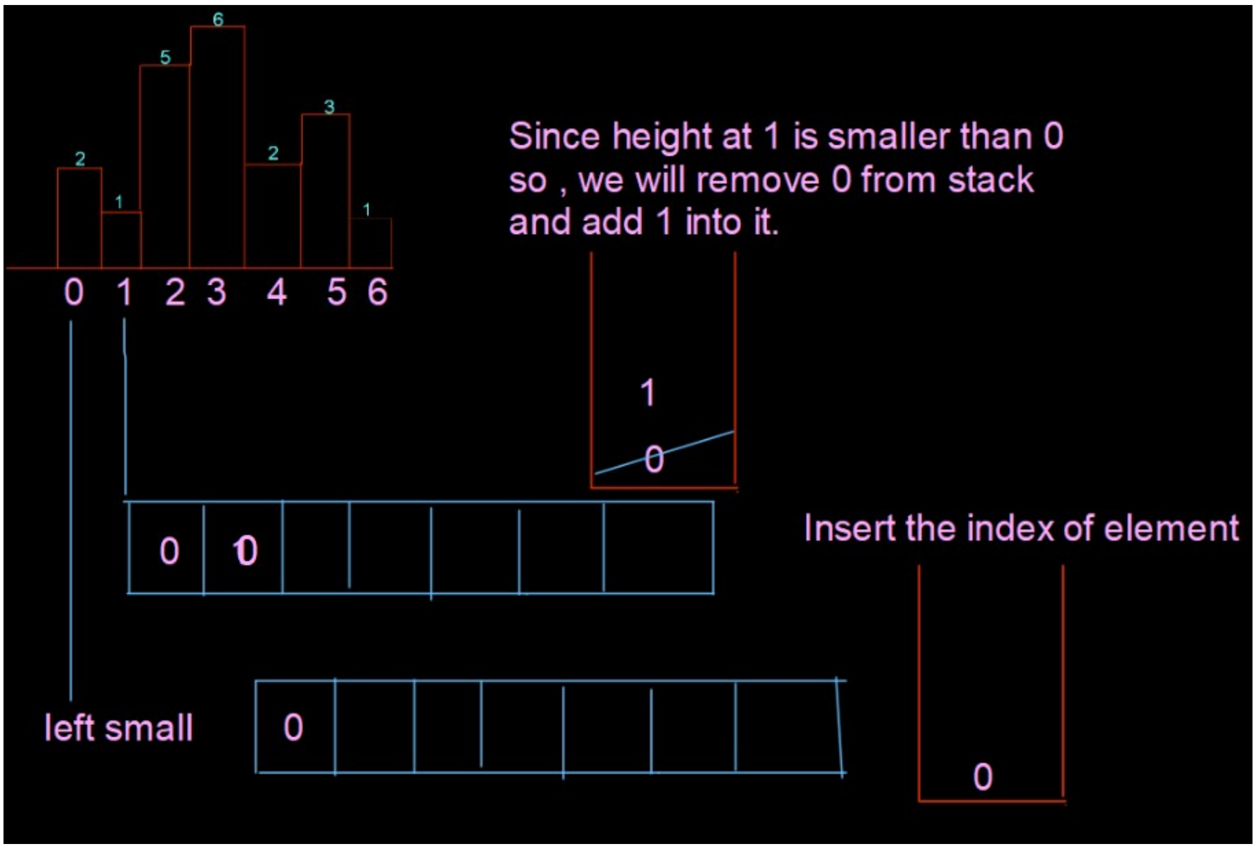
Space Complexity: O(1)

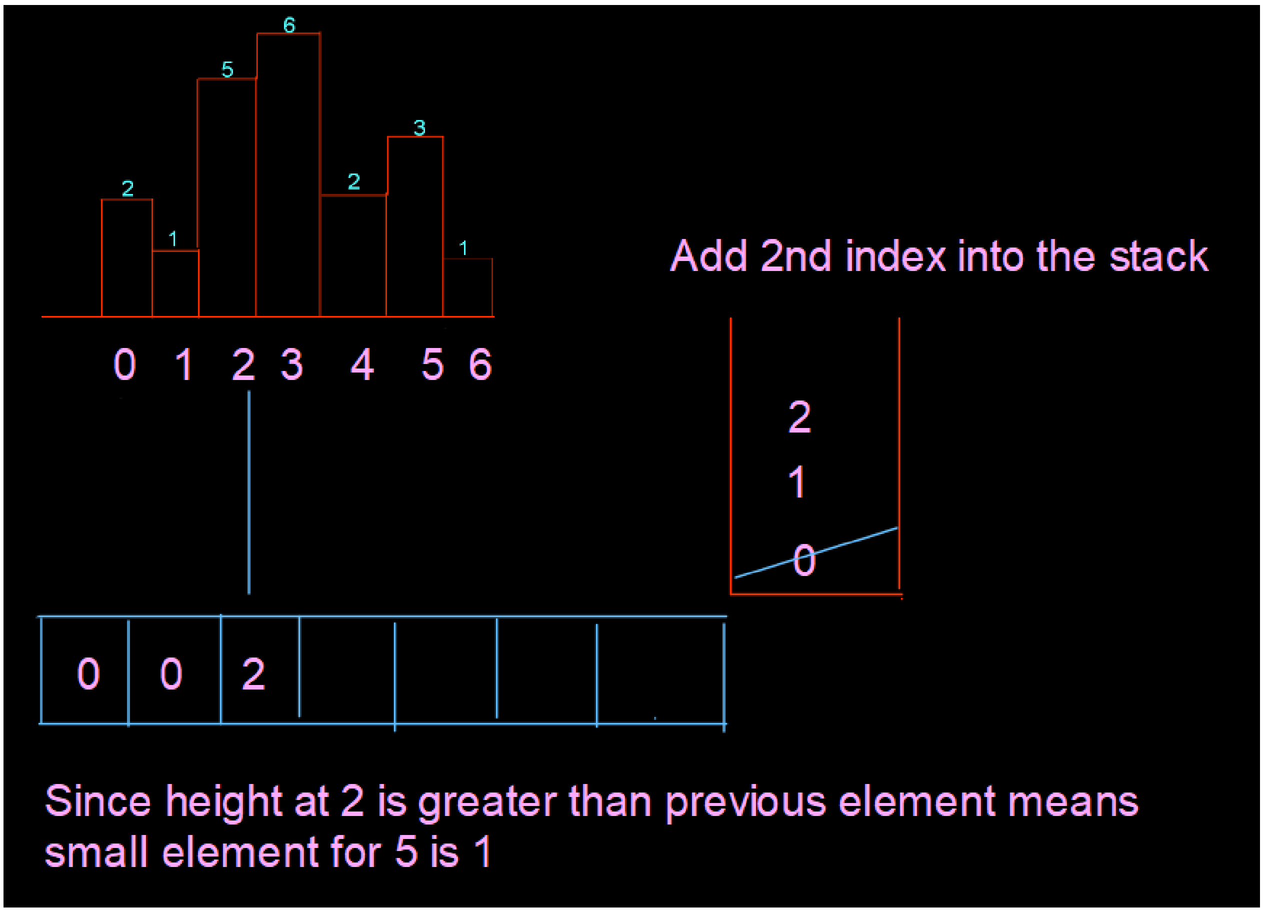
Solution 2: Optimised Approach 1

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.

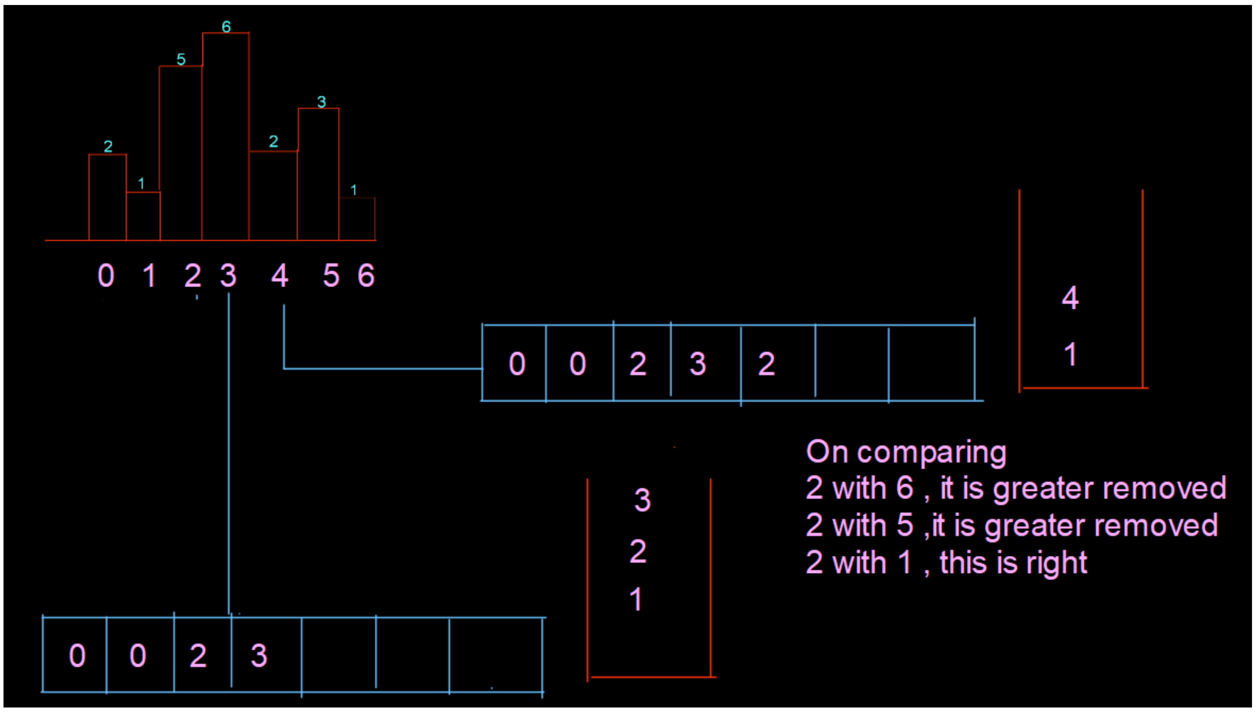
Approach:

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

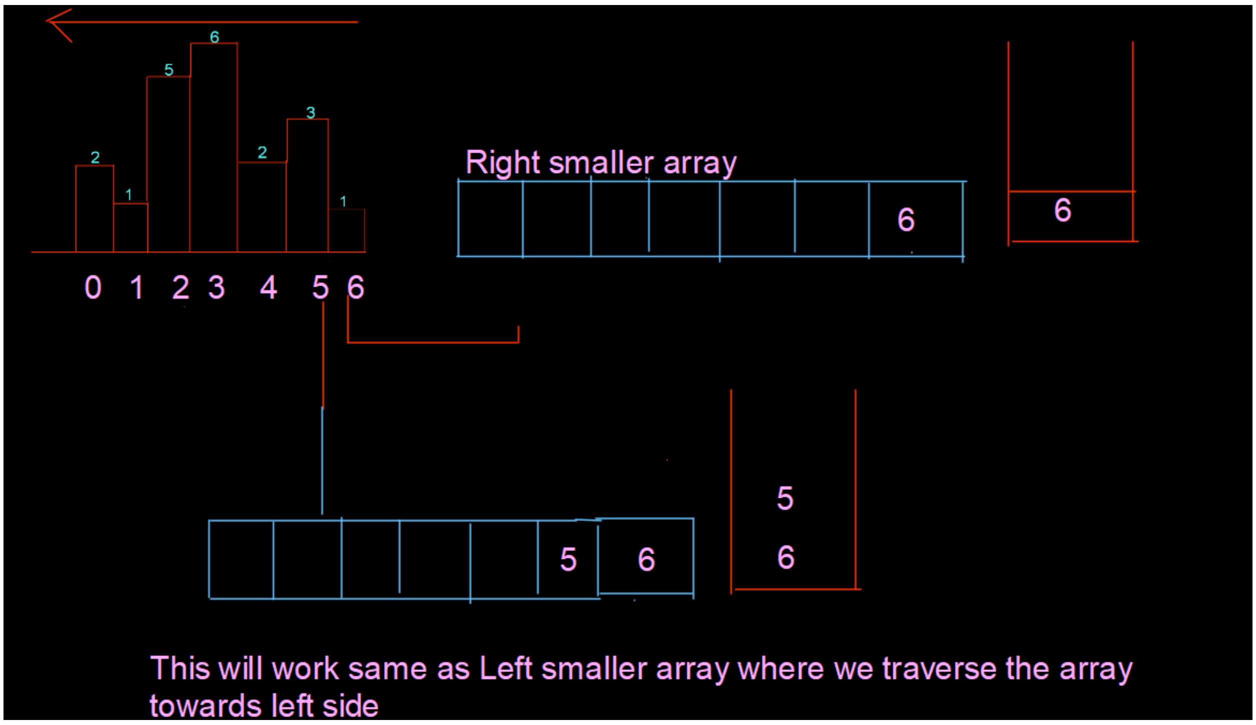

2
1
5
6
2
3
1
0 1 2 3 4 5 6
Since height at 1 is smaller than 0 so , we will remove 0 from stack and add 1 into it.
1
0
0
0
Insert the index of element
left small
0
0



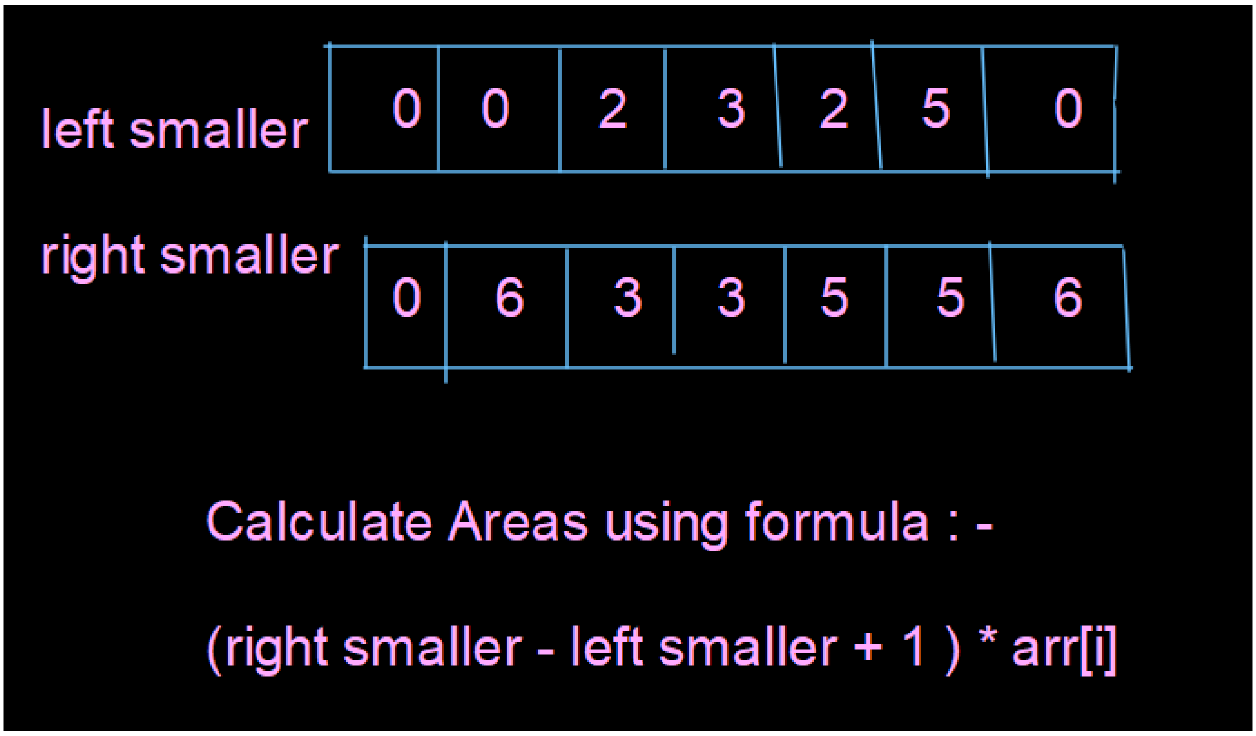
6
5
2
1
3
2
1
Add 2nd index into the stack
0 1 2 3 4 5 6
2
1
~~0~~
0
0
2
Since height at 2 is greater than previous element means small element for 5 is 1



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


2
1
5
6
2
3
1
0 1 2 3 4 5 6
Right smaller array
6
6
5 6
5
6
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:

```
import java.util.*;
// Brute Force Approach to find largest rectangle area in Histogram
public class Main {
    public static int largestRectangleArea(int[] heights) {
        int n = heights.length;
        Stack < Integer > st = new Stack < > ();
        int leftSmall[] = new int[n];
        int rightSmall[] = new int[n];
        for (int i = 0; i < n; i++) {
            while (!st.isEmpty() && heights[st.peek()] >= heights[i]) {
                st.pop();
            }

            if (st.isEmpty()) leftSmall[i] = 0;
            else leftSmall[i] = st.peek() + 1;
            st.push(i);
        }

        // clear the stack to be re-used
        while (!st.isEmpty()) st.pop();

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

            if (st.isEmpty()) rightSmall[i] = n - 1;
            else rightSmall[i] = st.peek() - 1;

            st.push(i);
        }

        int maxA = 0;
        for (int i = 0; i < n; i++) {
            maxA = Math.max(maxA, heights[i] * (rightSmall[i] - leftSmall[i] + 
1));
        }
        return maxA;

    }
    public static void main(String args[]) {
        int arr[] = {2, 1, 5, 6, 2, 3, 1};
        int n = 7;
        System.out.println("The largest area in the histogram is " + 
        largestRectangleArea(arr));

    }
}
```

Output: The largest area in the histogram is 10

Time Complexity: O(N)

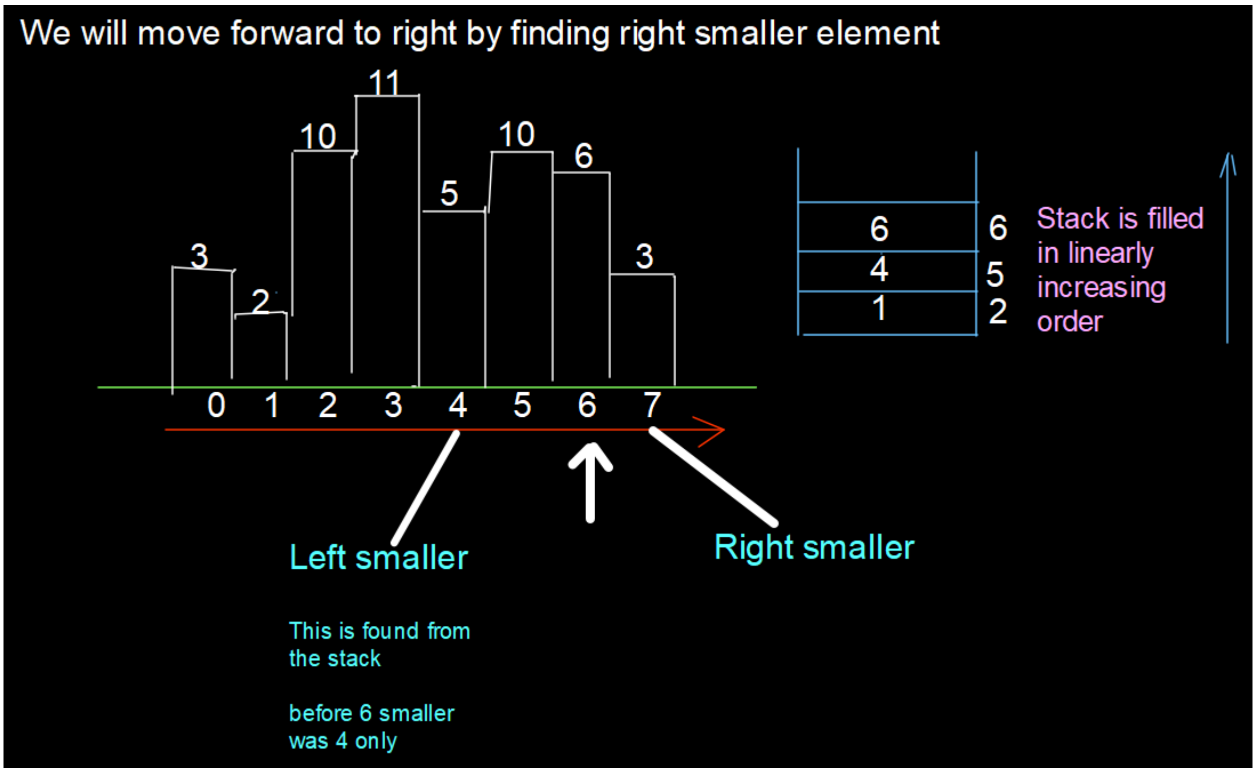
Space Complexity: O(3N) where 3 is for the stack, left small array and a right small array

Solution 3: Optimised approach 2

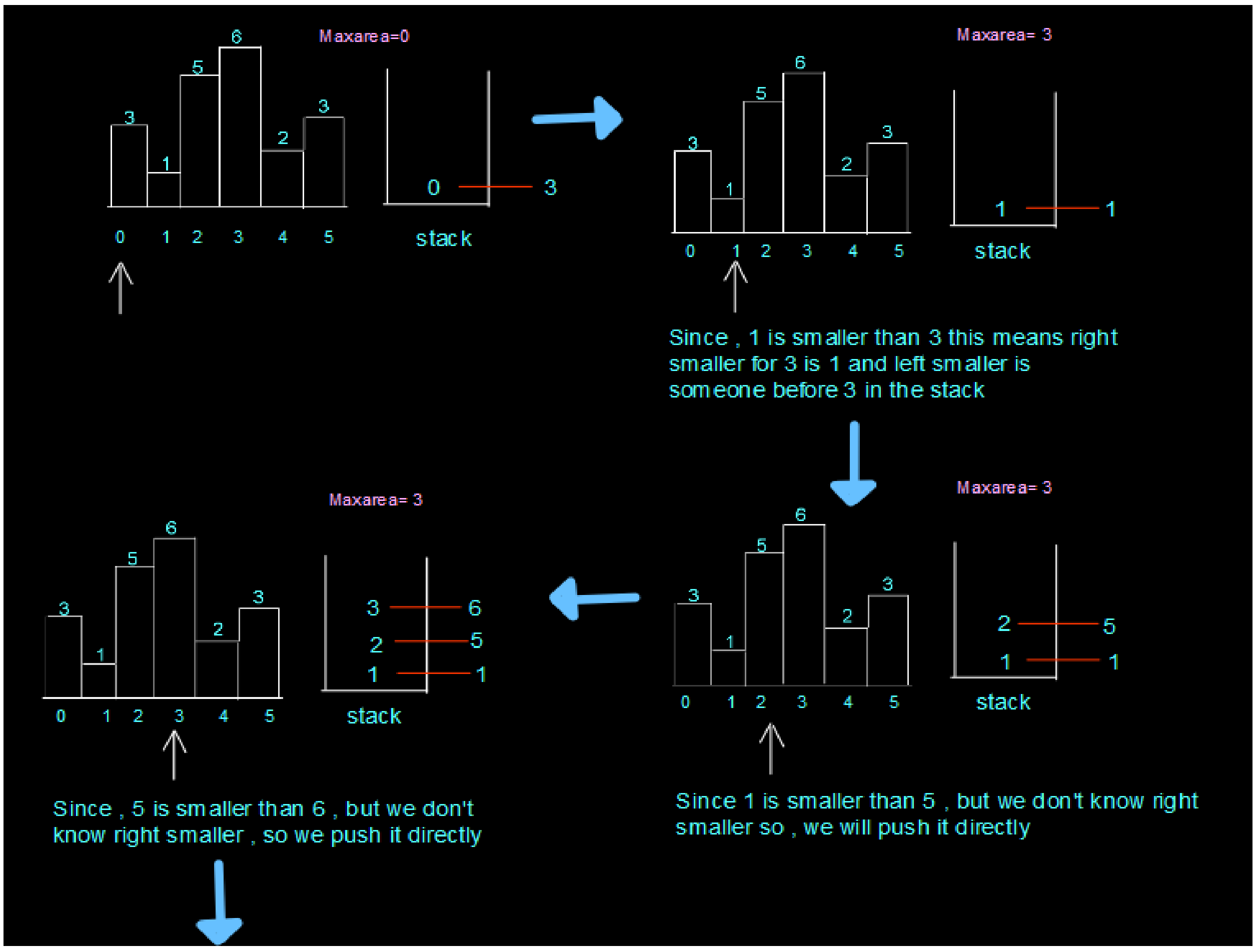
Intuition:

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
3
2
10
11
5
10
6
3
0 1 2 3 4 5 6 7
6
4
1
6
5
2
Stack is filled in linearly increasing order
Left smaller
This is found from the stack
before 6 smaller was 4 only
Right smaller

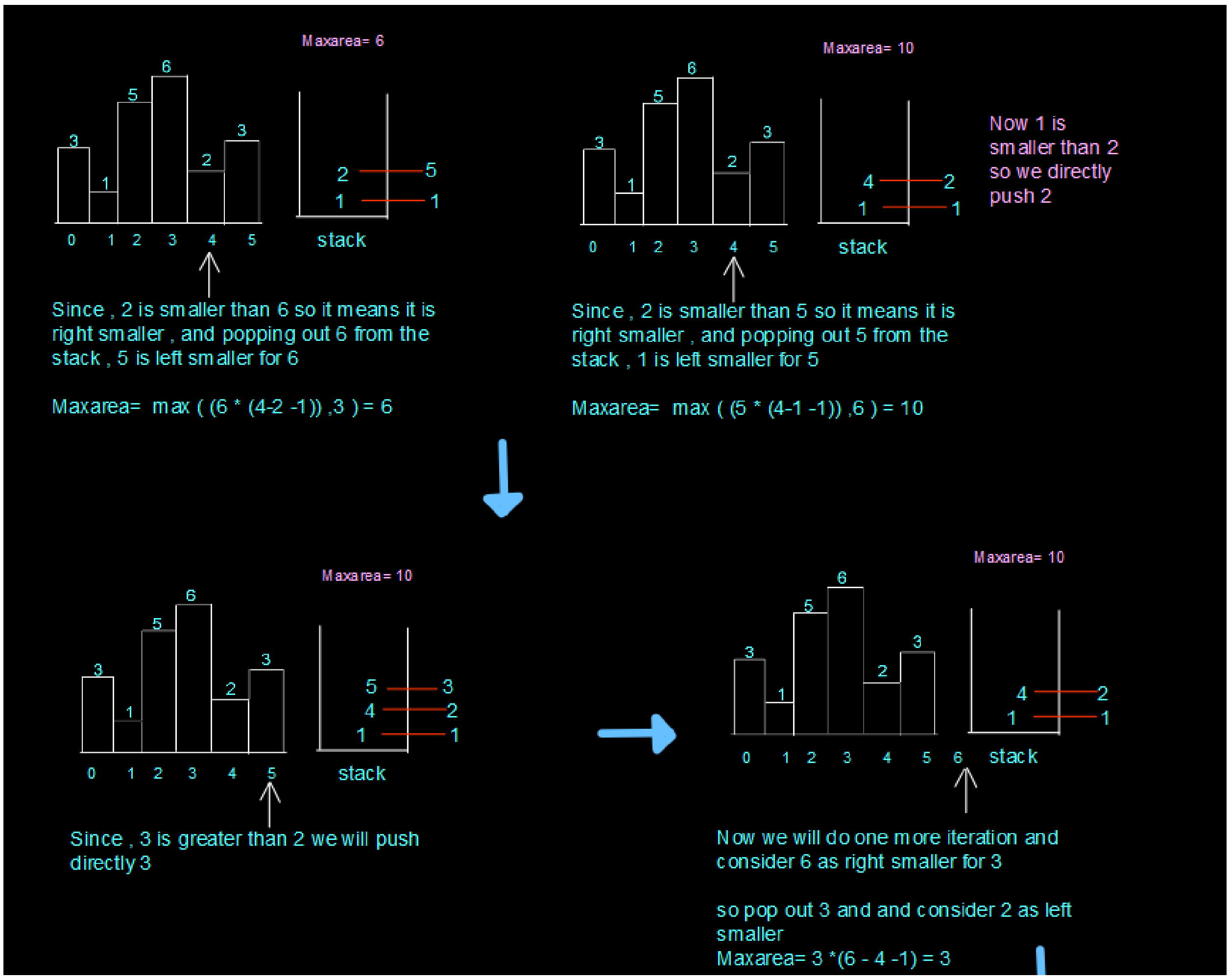
Approach:



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

Now 1 is smaller than 2 so we directly push 2

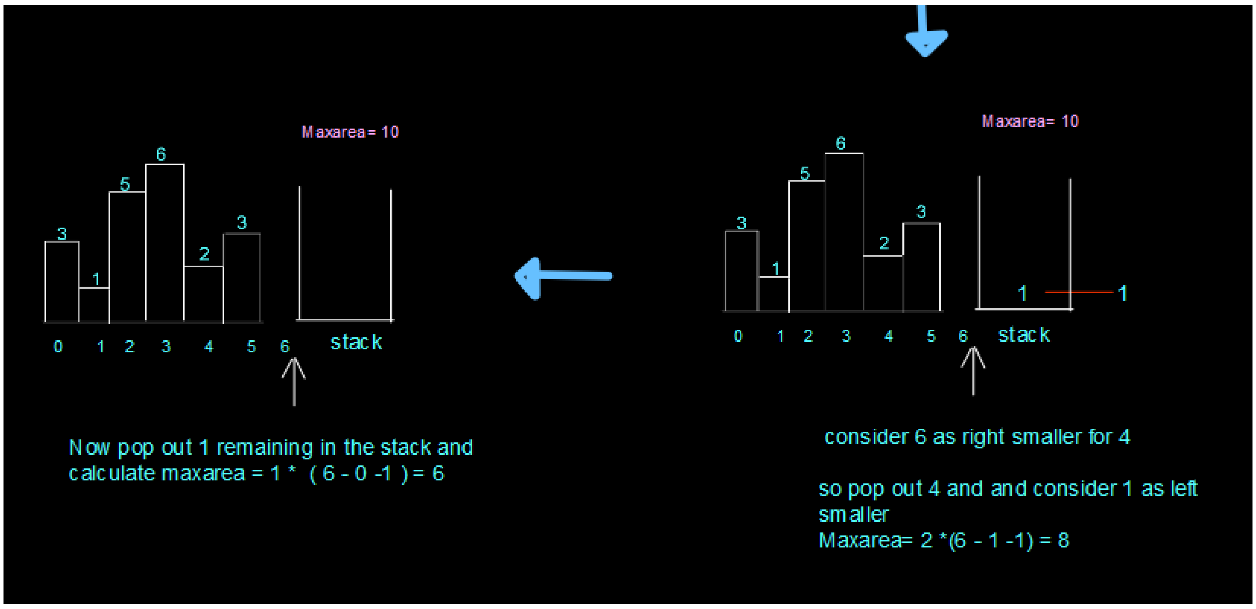
Since , 2 is smaller than 5 so it means it is right smaller , and popping out 5 from the stack , 1 is left smaller for 5

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

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


Maxarea= 10
3
1
5
6
2
3
0 1 2 3 4 5 6
stack
Now pop out 1 remaining in the stack and calculate maxarea = 1 * (6 - 0 -1) = 6
Maxarea= 10
3
1
5
6
2
3
1
1
0 1 2 3 4 5 6
stack
consider 6 as right smaller for 4
so pop out 4 and and consider 1 as left smaller
Maxarea= 2 *(6 - 1 -1) = 8

Code:

```java
import java.util.*;
public class Main {
    static int largestRectangleArea(int histo[]) {
        Stack < Integer > st = new Stack < > ();
        int maxA = 0;
        int n = histo.length;
        for (int i = 0; i <= n; i++) {
            while (!st.empty() && (i == n || histo[st.peek()] >= histo[i])) {
                int height = histo[st.peek()];
                st.pop();
                int width;
                if (st.empty())
                    width = i;
                else
                    width = i - st.peek() - 1;
                maxA = Math.max(maxA, width * height);
            }
            st.push(i);
        }
        return maxA;
    }

    public static void main(String args[]) {
        int histo[] = {3, 1, 5, 6, 2, 3};
        System.out.println("The largest area in the histogram is " +
largestRectangleArea(histo));
    }
}
```

Output: The largest area in the histogram is 10

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

Space Complexity: O(N)