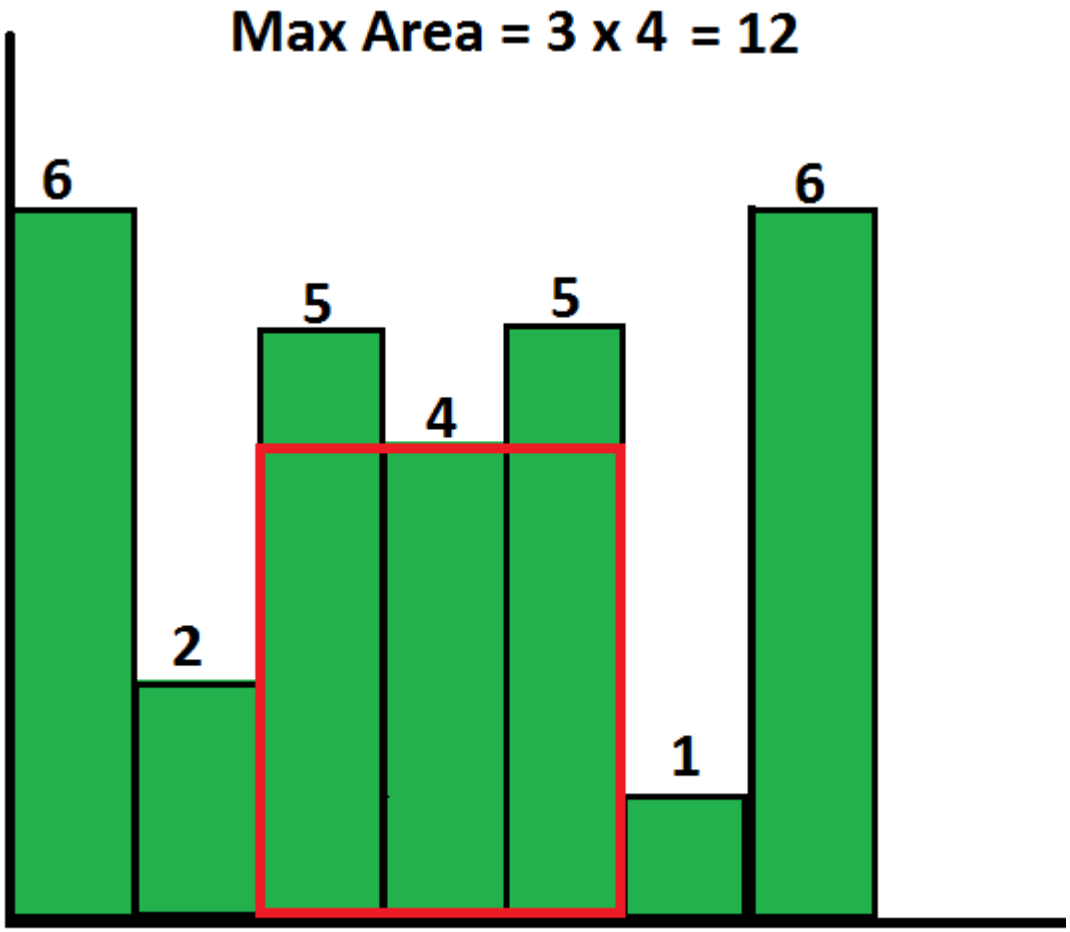
Dash
All
Articles
Videos
Problems
Quiz
Prev
Next

# Largest Rectangular Area in a Histogram

Find the largest rectangular area possible in a given histogram where the largest rectangle can be made of a number of contiguous bars whose heights are given in an array. For simplicity, assume that all bars have the same width and the width is 1 unit.

**Example:**

> **Input:** histogram = {6, 2, 5, 4, 5, 1, 6}




**Output:** 12

**Input:** histogram = {3, 5, 1, 7, 5, 9}
**Output:** 15

To solve the problem follow the below idea:

> For every bar 'x', we calculate the area with 'x' as the smallest bar in the rectangle. If we calculate the such area for every bar 'x' and find the maximum of all areas, our task is done.
>
> **How to calculate the area with 'x' as the smallest bar?**
>
> We need to know the index of the first smaller (smaller than 'x') bar on the left of 'x' and the index of the first smaller bar on the right of 'x'. Let us call these indexes as 'left index' and 'right index' respectively. We traverse all bars from left to right and maintain a stack of bars. Every bar is pushed to stack once. A bar is popped from the stack when a bar of smaller height is seen. When a bar is popped, we calculate the area with the popped bar as the smallest bar.
>
> **How do we get the left and right indexes of the popped bar?**
>
> The current index tells us the right index and the index of the previous item in the stack is the left index

Follow the given steps to solve the problem:

1. Create an empty stack.
2. Start from the first bar, and do the following for every bar hist[i] where '**i**' varies from 0 to n-1
    1. If the stack is empty or hist[i] is higher than the bar at top of the stack, then push '**i**' to stack.
    2. If this bar is smaller than the top of the stack, then keep removing the top of the stack while the top of the stack is greater.
    3. Let the removed bar be hist[tp]. Calculate the area of the rectangle with hist[tp] as the smallest bar.
    4. For hist[tp], the 'left index' is previous (previous to tp) item in stack and 'right index' is '**i**' (current index).
3. If the stack is not empty, then one by one remove all bars from the stack and do step (2.2 and 2.3) for every removed bar

Below is the implementation of the above approach.

C++ Java



```java
// Java program to find maximum rectangular area in linear
// time

import java.util.Stack;

public class RectArea {
    // The main function to find the maximum rectangular
    // area under given histogram with n bars
    static int getMaxArea(int hist[], int n)
    {
        // Create an empty stack. The stack holds indexes of
        // hist[] array The bars stored in stack are always
        // in increasing order of their heights.
        Stack<Integer> s = new Stack<>();

        int max_area = 0; // Initialize max area
        int tp; // To store top of stack
        int area_with_top; // To store area with top bar as
                           // the smallest bar

        // Run through all bars of given histogram
        int i = 0;
        while (i < n) {
            // If this bar is higher than the bar on top
            // stack, push it to stack
            if (s.empty() || hist[s.peek()] <= hist[i])
                s.push(i++);
```

```
        // If this bar is lower than top of stack, then
        // calculate area of rectangle with stack top as
        // the smallest (or minimum height) bar. 'i' is
        // 'right index' for the top and element before
        // top in stack is 'left index'
        else {
            tp = s.peek(); // store the top index
            s.pop(); // pop the top

            // Calculate the area with hist[tp] stack as
            // smallest bar
            area_with_top
                = hist[tp]
                  * (s.empty() ? i : i - s.peek() - 1);

            // update max area, if needed
            if (max_area < area_with_top)
                max_area = area_with_top;
        }
    }

    // Now pop the remaining bars from stack and
    // calculate area with every popped bar as the
    // smallest bar
    while (s.empty() == false) {
        tp = s.peek();
        s.pop();
        area_with_top
```

```
                = hist[tp]
                  * (s.empty() ? i : i - s.peek() - 1);

            if (max_area < area_with_top)
                max_area = area_with_top;
        }

        return max_area;
    }

    // Driver code
    public static void main(String[] args)
    {
        int hist[] = { 6, 2, 5, 4, 5, 1, 6 };

        // Function call
        System.out.println("Maximum area is "
                           + getMaxArea(hist, hist.length));
    }
}
```

**Output**

```
Maximum area is 12
```

**Time Complexity:** O(N), Since every bar is pushed and popped only once
**Auxiliary Space:** O(N)

## Largest Rectangular Area in a Histogram by finding the next and the previous smaller element:

To solve the problem follow the below idea:

> Find the previous and the next smaller element for every element of the histogram, as this would help to calculate the length of the subarray in which this current element is the minimum element. So we can create a rectangle of size **(current element * length of the subarray)** using this element. Take the maximum of all such rectangles.

Follow the given steps to solve the problem:

- First, we will take two arrays **left_smaller[]** and **right_smaller[]** and initialize them with -1 and n respectively
- For every element, we will store the index of the previous smaller and next smaller element in left_smaller[] and right_smaller[] arrays respectively
- Now for every element, we will calculate the area by taking this $i^{th}$ element as the smallest in the range left_smaller[i] and right_smaller[i] and multiplying it with the difference of left_smaller[i] and right_smaller[i]
- We can find the maximum of all the areas calculated in step 3 to get the desired maximum area

Below is the implementation of the above approach.

C++ Java

```
// Java code for the above approach

import java.io.*;
import java.lang.*;
import java.util.*;
```

```
public class RectArea {

    // Function to find largest rectangular area possible in
    // a given histogram.
    public static int getMaxArea(int arr[], int n)
    {
        // your code here
        // we create an empty stack here.
        Stack<Integer> s = new Stack<>();
        // we push -1 to the stack because for some elements
        // there will be no previous smaller element in the
        // array and we can store -1 as the index for
        // previous smaller.
        s.push(-1);
        int max_area = arr[0];
        // We declare left_smaller and right_smaller array
        // of size n and initialize them with -1 and n as
        // their default value. left_smaller[i] will store
        // the index of previous smaller element for ith
        // element of the array. right_smaller[i] will store
        // the index of next smaller element for ith element
        // of the array.
        int left_smaller[] = new int[n];
        int right_smaller[] = new int[n];
        for (int i = 0; i < n; i++) {
            left_smaller[i] = -1;
            right_smaller[i] = n;
        }
```

```
int i = 0;
while (i < n) {
    while (!s.empty() && s.peek() != -1
           && arr[i] < arr[s.peek()]) {
        // if the current element is smaller than
        // element with index stored on the top of
        // stack then, we pop the top element and
        // store the current element index as the
        // right_smaller for the popped element.
        right_smaller[s.peek()] = (int)i;
        s.pop();
    }
    if (i > 0 && arr[i] == arr[(i - 1)]) {
        // we use this condition to avoid the
        // is same as current element, the
        // left_smaller will always be the same for
        // both.
        left_smaller[i]
            = left_smaller[(int)(i - 1)];
    }
    else {
        // Element with the index stored on the top
        // of the stack is always smaller than the
        // current element. Therefore the
        // left_smaller[i] will always be s.top().
        left_smaller[i] = s.peek();
    }
```

```
            s.push(i);
            i++;
        }

        for (i = 0; i < n; i++) {
            // here we find area with every element as the
            // smallest element in their range and compare
            // it with the previous area. in this way we get
            // our max Area form this.
            max_area = Math.max(
                max_area, arr[i]
                              * (right_smaller[i]
                                 - left_smaller[i] - 1));
        }

        return max_area;
    }

    // Driver code
    public static void main(String[] args)
    {
        int hist[] = { 6, 2, 5, 4, 5, 1, 6 };

        // Function call
        System.out.println("Maximum area is "
                           + getMaxArea(hist, hist.length));
    }
}
```

Dash
All
Articles
Videos
Problems
Quiz

## Output

```
maxArea = 12
```

**Time Complexity:** O(N)
**Auxiliary Space:** O(N)