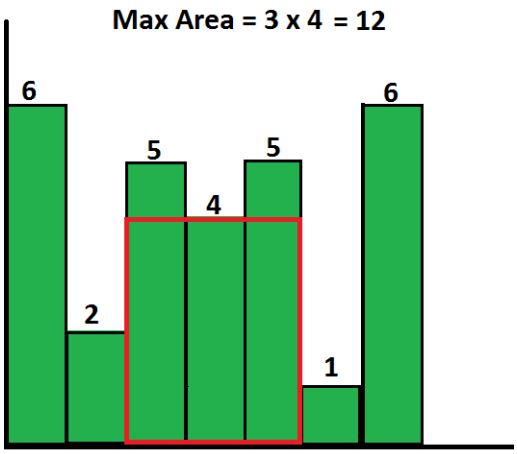
Largest Rectangular Area in a Histogram | Set 2

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

For example, consider the following histogram with 7 bars of heights {6, 2, 5, 4, 5, 2, 6}. The largest possible rectangle possible is 12 (see the below figure, the max area rectangle is highlighted in red)



We strongly recommend that you click here and practice it, before moving on to the solution.

We have discussed a Divide and Conquer based O(nLogn) solution for this problem. In this post, O(n) time solution is discussed. Like the previous post, width of all bars is assumed to be 1 for simplicity. For every bar 'x', we calculate the area with 'x' as the smallest bar in the rectangle. If we calculate such area for every bar 'x' and find the maximum of all areas, our task is done. How to calculate area with 'x' as smallest bar? We need to know index of the first smaller (smaller than 'x') bar on left of 'x' and index of first smaller bar on right of 'x'. Let us call these indexes as 'left index' and 'right index' respectively.
We traverse all bars from left to right, maintain a stack of bars. Every bar is pushed to stack once. A bar is popped from stack when a bar of smaller height is seen. When a bar is popped, we calculate the area with the popped bar as smallest bar. How do we get left and right indexes of the popped bar – the current index tells us the 'right index' and index of previous item in stack is the 'left index'. Following is the complete algorithm.

1) Create an empty stack.

2) Start from first bar, and do following for every bar 'hist[i]' where 'i' varies from 0 to n-1.
.....**a)** If stack is empty or hist[i] is higher than the bar at top of stack, then push 'i' to stack.
.....**b)** If this bar is smaller than the top of stack, then keep removing the top of stack while top of the stack is greater. Let the removed bar be hist[tp]. Calculate area of rectangle with hist[tp] as smallest bar. 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 stack and do step 2.b for every removed bar.

Following is C++ implementation of the above algorithm.

```
// C++ program to find maximum rectangular area in linear time
#include<iostream>
#include<stack>
using namespace std;

// The main function to find the maximum rectangular area under given
// histogram with n bars
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<int> s;

    int max_area = 0; // Initalize 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.top()] <= 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.top();  // 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.top() - 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.top();
        s.pop();
        area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);

        if (max_area < area_with_top)
            max_area = area_with_top;
    }

    return max_area;
}

// Driver program to test above function
int main()
{
    int hist[] = {6, 2, 5, 4, 5, 1, 6};
    int n = sizeof(hist)/sizeof(hist[0]);
    cout << "Maximum area is " << getMaxArea(hist, n);
    return 0;
}
```

Output:

```
Maximum area is 12
```

Time Complexity: Since every bar is pushed and popped only once, the time complexity of this method is O(n).

References

http://www.informatik.uni-ulm.de/acm/Locals/2003/html/histogram.html

http://www.informatik.uni-ulm.de/acm/Locals/2003/html/judge.html