GetAhead - Interview Practice 4

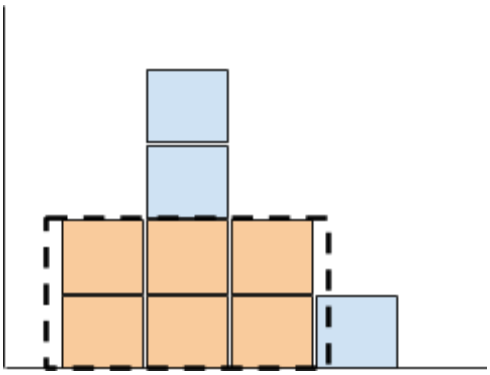
Histograms and Areas

Given an array of non-negative integers that represent the bars (y value) in a histogram (with the array index being the x value), find the rectangle with the largest area under the curve and above the axis. Return the pair of array indices that represent the rectangle.

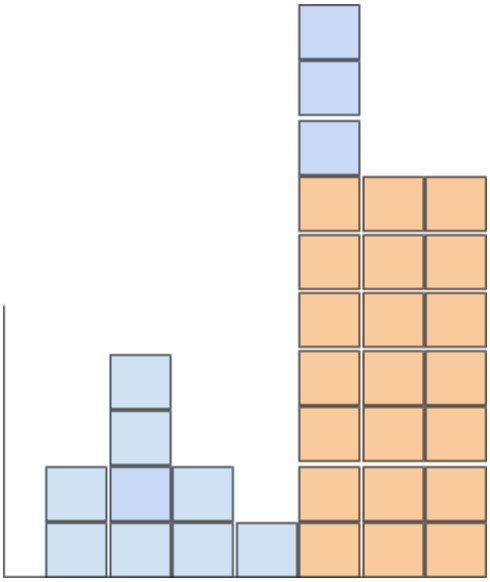
Test Cases

Note that there may be other valid answers.

For array `[2,4,2,1]`, the largest area is 6, with height 2, and width from indices 0 to 2:



For array `[2,4,2,1,10,6,10]` the largest area is 18, with height 6 and width from indices 4 to 6.



Solution

JAVA

```java
// JAVA SOLUTION
import java.util.Stack;
// This is a solution for the Histograms and Areas problem.
// It is partially taken from
https://www.geeksforgeeks.org/largest-rectangle-under-histogram/

public class FindMaxRectangle {
    private int histogram[];
    private int histogramLen;
    private Stack<Integer> stack;
    private int max_area = 0; // Initialize max area
    private int left_index = 0; // The left index of the max area rectangle
    private int right_index = 0; // The right index of the max area rectangle


    public FindMaxRectangle(int histogram[]) {
        this.histogram = histogram;
        this.histogramLen = histogram.length;
        this.stack = new Stack<>();
    }

    public int getRightIndex() {
        return this.right_index;
    }
```

```
    public int getLeftIndex() {
        return this.left_index;
    }

    private void calculateAndUpdateArea(int top, int current_position) {
        int right_position = current_position - 1;
        int left_position = stack.empty() ? 0 : stack.peek() + 1;
        int rect_width = right_position - left_position + 1;
        int rect_height = histogram[top];
        int area = rect_height * rect_width;
        if (max_area < area) {
            max_area = area;
            left_index = left_position;
            right_index = right_position;
        }
    }

    public int findMaxRectangle() {
        int top; // To store top of stack

        // Run through all bars of given histogram
        for (int current_position = 0; current_position < histogramLen;
current_position++) {

            // If this bar is lower than top of stack, then calculate area of
rectangle
            // with stack top as the smallest (or minimum height) bar.
            while (!this.stack.empty() && this.histogram[this.stack.peek()] >=
this.histogram[current_position]) {
                top = this.stack.peek(); // store the top index
                this.stack.pop(); // pop the top
                // Calculate the area with hist[top] stack as smallest bar
                // update max area, if needed
                calculateAndUpdateArea(top, current_position);
            }
            this.stack.push(current_position);
        }

        // Now pop the remaining bars from stack and calculate area with every
popped bar as the smallest bar
        while (!this.stack.empty()) {
            top = this.stack.peek();
            this.stack.pop();
            calculateAndUpdateArea(top, histogramLen);
        }

        return max_area;
```

```
    }

    public static void main(String[] args) {
        int hist1[] = {2, 4, 2, 1};
        FindMaxRectangle finder1 = new FindMaxRectangle(hist1);
        int maxRectangle = finder1.findMaxRectangle();
        System.out.println("The maximum area is " + maxRectangle + " with indices "
+ finder1.getLeftIndex() + " " + finder1.getRightIndex());

        int hist2[] = {2, 4, 2, 1, 10, 6, 10};
        FindMaxRectangle finder2 = new FindMaxRectangle(hist2);
        int maxRectangle2 = finder2.findMaxRectangle();
        System.out.println("The maximum area is " + maxRectangle2 + " with indices "
+ finder2.getLeftIndex() + " " + finder2.getRightIndex());

    }

}
```

C++

```
// C++ Solution

#include <cassert>
#include <stack>
#include <utility>
#include <vector>

class LargestRectangleCalculator {
 public:
  LargestRectangleCalculator(std::vector<int> histogram)
      : histogram_(histogram) {}
  ~LargestRectangleCalculator() = default;

  std::pair<int, int> GetLargestRectangularAreaUnderCurve() {
    CalculateLargestRectangle();
    return std::pair<int, int>(low_index_, high_index_);
  }

  void PushCurrentPosOnStack(int current_pos) {
    positions_.push(current_pos);
  }

  void PopFromStacksAndUpdateMaxArea(int current_pos) {
```

```
    assert((!positions_.empty()));

    // The largest rectangle so far with the height corresponding to the top
    // of stack will have its left index as the index right after the second to
    // top element in the stack. Its right index will be right before the
    // the current position.
    int top_position = positions_.top();
    positions_.pop();

    int right_index = current_pos -1;
    int left_index = positions_.empty() ? 0: positions_.top() + 1;
    int rectangle_width = right_index - left_index + 1;
    int rectangle_height = histogram_[top_position];

    if (rectangle_width * rectangle_height > max_area_) {
      max_area_ = rectangle_width * rectangle_height;
      low_index_ = left_index;
      high_index_ = right_index;
    }
  }

  void CalculateLargestRectangle() {
    if (histogram_.empty()) return;

    for (int current_pos = 0; current_pos < histogram_.size(); current_pos++) {
      while (!positions_.empty() && histogram_[positions_.top()] >=
histogram_[current_pos]) {
        PopFromStacksAndUpdateMaxArea(current_pos);
      }
      PushCurrentPosOnStack(current_pos);
    }
    // Pop all remaining values from the stacks.
    while (!positions_.empty()) PopFromStacksAndUpdateMaxArea(histogram_.size());
  }

 private:
  std::vector<int> histogram_;
  std::stack<int> positions_;
  int max_area_ = -1;
  int low_index_ = -1;
  int high_index_ = -1;
};

int main(int argc, char** argv) {
  LargestRectangleCalculator calculator_1(std::vector<int>({}));
  assert((calculator_1.GetLargestRectangularAreaUnderCurve() ==
          std::pair<int, int>(-1, -1)));
```

```
  LargestRectangleCalculator calculator_2(std::vector<int>({2, 4, 2, 1}));
  assert((calculator_2.GetLargestRectangularAreaUnderCurve() ==
         std::pair<int, int>(0, 2)));

  LargestRectangleCalculator calculator_3(
      std::vector<int>({2, 4, 2, 1, 10, 6, 10}));
  assert((calculator_3.GetLargestRectangularAreaUnderCurve() ==
         std::pair<int, int>(4, 6)));

  LargestRectangleCalculator calculator_4(std::vector<int>({2, 2, 2, 2}));
  assert((calculator_4.GetLargestRectangularAreaUnderCurve() ==
         std::pair<int, int>(0, 3)));

  LargestRectangleCalculator calculator_5(std::vector<int>({8,0,1,1}));
  assert((calculator_5.GetLargestRectangularAreaUnderCurve() ==
         std::pair<int, int>(0, 0)));
  return 0;
}
```

Python

```
See cl/246763238.
```