maximumareainhistogram\Solution.java

```java
package maximumareainhistogram;

import java.util.Stack;

public class Solution {

    /**
     * Given an array of integers representing the heights of a histogram, this
     * function calculates the maximum area of a rectangle that can be inscribed
     * within the histogram.
     *
     * @param heights an array of integers representing the heights of a
     * histogram
     * @return the maximum area of a rectangle that can be inscribed within the
     *
     * histogram
     */

    public static int maxAreaRectangle(int[] heights){
        int maxArea = 0;
        int nextSmallerElementRight[] = nextSmallerRight(heights);
        int nextSmallerElementLeft[] = nextSmallerLeft(heights);
        for (int i = 0; i < heights.length; i++) {
            int height = heights[i];
            int width = nextSmallerElementRight[i]-nextSmallerElementLeft[i]-1;
            int area = height*width;
            if(maxArea < area){
                maxArea = area;
            }
        }
        return maxArea;
    }
    /**
     * Given an array of integers representing the heights of a histogram, this
     * function returns an array of the same length, where the value at each index
     * is the index of the next smaller element to the left of the element at that
     * index. If there is no such element, the value at that index is -1.
     *
     * @param heights an array of integers representing the heights of a
     * histogram
     * @return an array of the same length, where the value at each index is the
     * index of the next smaller element to the left of the element at that index
     */
    public static int[] nextSmallerLeft(int[] heights){
        Stack<Integer> s = new Stack<>();
        int nextSmallerLeft[] = new int[heights.length];
        for (int i = 0; i < heights.length; i++) {
            while (!s.isEmpty() && heights[s.peek()] >= heights[i]) {
                s.pop();
            }
            if (s.isEmpty()) {
                nextSmallerLeft[i] = -1;
            }else{
                nextSmallerLeft[i] = s.peek();
            }
            s.push(i);
        }
        return nextSmallerLeft;
    }
    /**
     * Given an array of integers representing the heights of a histogram, this
     * function returns an array of the same length, where the value at each index
     * is the index of the next smaller element to the right of the element at that
     * index. If there is no such element, the value at that index is -1.
     *
     * @param heights an array of integers representing the heights of a
     * histogram
     * @return an array of the same length, where the value at each index is the
     * index of the next smaller element to the right of the element at that index
     */
    public static int[] nextSmallerRight(int[] heights){
        Stack<Integer> s = new Stack<>();
        int nextSmallerRight[] = new int[heights.length];
        for (int i = heights.length-1; i >= 0; i--) {
            while (!s.isEmpty() && heights[s.peek()] >= heights[i]) {
                s.pop();
            }
            if (s.isEmpty()) {
                nextSmallerRight[i] = -1;
            }else{
                nextSmallerRight[i] = s.peek();
            }
            s.push(i);
        }
        return nextSmallerRight;
    }
    public static void main(String[] args) {
        int heights[] = {2,4};
        System.out.println(maxAreaRectangle(heights));
    }
}
```