

maximumareainhistogram\Solution.java

```
1 package maximumareainhistogram;
2
3 import java.util.Stack;
4
5 public class Solution {
6
7     /**
8      * Given an array of integers representing the heights of a histogram, this
9      * function calculates the maximum area of a rectangle that can be inscribed
10     * within the histogram.
11     *
12     * @param heights an array of integers representing the heights of a
13     * histogram
14     * @return the maximum area of a rectangle that can be inscribed within the
15     *
16     * histogram
17     */
18
19     public static int maxAreaRectangle(int[] heights){
20         int maxArea = 0;
21         int nextSmallerElementRight[] = nextSmallerRight(heights);
22         int nextSmallerElementLeft[] = nextSmallerLeft(heights);
23         for (int i = 0; i < heights.length; i++) {
24             int height = heights[i];
25             int width = nextSmallerElementRight[i]-nextSmallerElementLeft[i]-1;
26             int area = height*width;
27             if(maxArea < area){
28                 maxArea = area;
29             }
30         }
31         return maxArea;
32     }
33     /**
34     * Given an array of integers representing the heights of a histogram, this
35     * function returns an array of the same length, where the value at each index
36     * is the index of the next smaller element to the left of the element at that
37     * index. If there is no such element, the value at that index is -1.
38     *
39     * @param heights an array of integers representing the heights of a
40     * histogram
41     * @return an array of the same length, where the value at each index is the
42     * index of the next smaller element to the left of the element at that index
43     */
44     public static int[] nextSmallerLeft(int[] heights){
45         Stack<Integer> s = new Stack<>();
46         int nextSmallerLeft[] = new int[heights.length];
47         for (int i = 0; i < heights.length; i++) {
48             while (!s.isEmpty() && heights[s.peek()] >= heights[i]) {
```

```

49         s.pop();
50     }
51     if (s.isEmpty()) {
52         nextSmallerLeft[i] = -1;
53     }else{
54         nextSmallerLeft[i] = s.peek();
55     }
56     s.push(i);
57 }
58 return nextSmallerLeft;
59 }
60 /**
61  * Given an array of integers representing the heights of a histogram, this
62  * function returns an array of the same length, where the value at each index
63  * is the index of the next smaller element to the right of the element at that
64  * index. If there is no such element, the value at that index is -1.
65  *
66  * @param heights an array of integers representing the heights of a
67  * histogram
68  * @return an array of the same length, where the value at each index is the
69  * index of the next smaller element to the right of the element at that index
70  */
71 public static int[] nextSmallerRight(int[] heights){
72     Stack<Integer> s = new Stack<>();
73     int nextSmallerRight[] = new int[heights.length];
74     for (int i = heights.length-1; i >= 0; i--) {
75         while (!s.isEmpty() && heights[s.peek()] >= heights[i]) {
76             s.pop();
77         }
78         if (s.isEmpty()) {
79             nextSmallerRight[i] = -1;
80         }else{
81             nextSmallerRight[i] = s.peek();
82         }
83         s.push(i);
84     }
85     return nextSmallerRight;
86 }
87 public static void main(String[] args) {
88     int heights[] = {2,4};
89     System.out.println(maxAreaRectangle(heights));
90 }
91 }
92

```