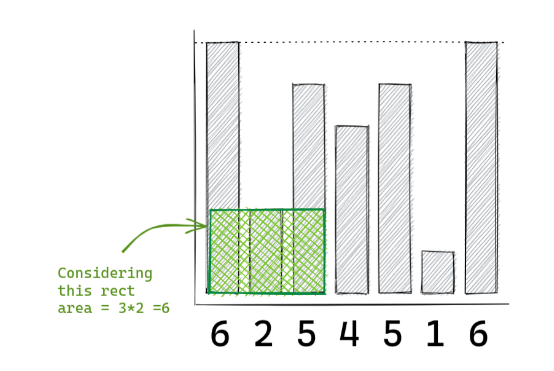
**1. Problem Discussion**

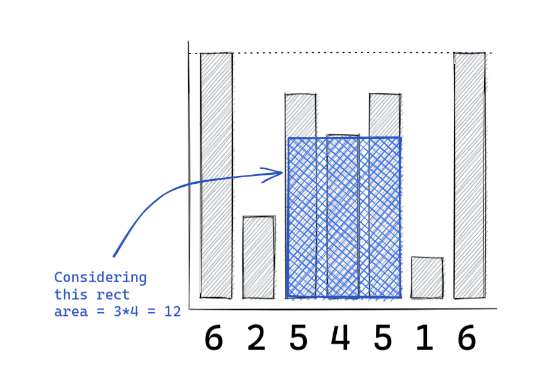
You will be given a number ‘n’ representing the size of the array ‘a’. Then, you will be given ‘n’ numbers which represent the height of bars in a bar chart. You have to find and print the area of the largest rectangle in the histogram.

**2. Approach**

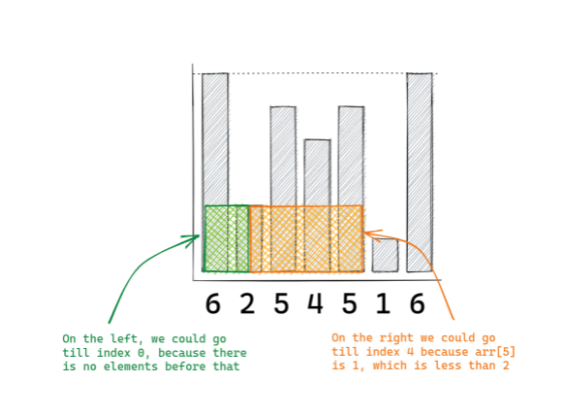
Let's take an array to understand the problem: arr[] = [6, 2, 5, 4, 5, 1, 6] If we consider the rectangle from 0th index to 2nd index, we will get a rectangle of height 2 and width 3, so, the area of that rectangle will be 3\*2 which is 6. You have to maximize this area.

****

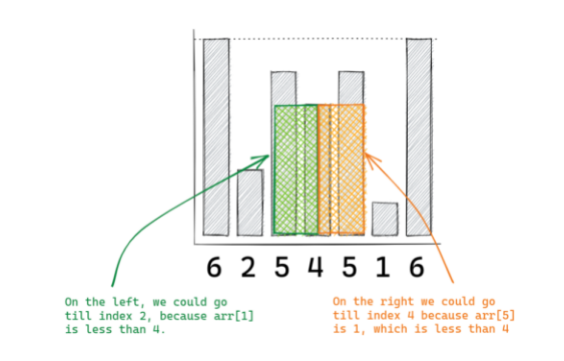
Now, examine the rectangle from 2nd index to 4th index with height 4 and width 2 to 4.

****

What we can do is have each number stretch as far to the left as possible and as far to the right as possible, so that the height of the bars is at least equal to the height of the current bar. So, if we consider the same example, and see the case for 1st index:

****

If we take the 3rd index as our current:

****

So, what exactly are these extremes on the left and right? These are nothing but the next smaller element to the right and the next smaller element to the left. The width of the rectangle is r-l-1 if the NSE to the right is r and the NSE to the left is l. In the illustration above, the NSE to the left is 1 and the NSE to the right is 5. As a result, 5-1-1 = 3, which is the rectangle's width. And the current element's height is the value. As a result, the area is 3\*4 = 12. But what if there is no next smaller element to the right and left? Consider the value at index 1. To the left, there are no smaller elements. For index 1, we'll suppose the NSE to the left is -1. Take, for example, the value at index 6, which has no NSE to the right. For this, the value will be arr.length, which in this case is 7. This will save us from having to deal with certain additional conditions. The code for NSE to the left and right is now known. To get the solution, combine that with the logic we discussed.

**3. Pseudo Code**

1. Iterate through the string and : a. Refer to the code for NSE index to the right and create an array. b. Refer to the code for NSE index to the left and create an array. c. Iterate through the given array. i. the width will be right index - left index + 1 ii. the height is a[index] iii. so the area is height\*width, if it's greater than the existing maxArea it becomes the new maxArea.

**4. Code**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

int[] arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = Integer.parseInt(br.readLine());

}

int[] rb = new int[arr.length]; // nse on the right

Stack<Integer> st = new Stack<>();

st.push(arr.length - 1);

rb[arr.length - 1] = arr.length;

for (int i = arr.length - 2; i >= 0; i--) {

while (st.size() > 0 && arr[i] <= arr[st.peek()]) {

st.pop();

}

if (st.size() == 0) {

rb[i] = arr.length;

} else {

rb[i] = st.peek();

}

st.push(i);

}

int[] lb = new int[arr.length]; // nse on the left

st = new Stack<>();

st.push(0);

lb[0] = -1;

for (int i = 1; i < arr.length; i++) {

while (st.size() > 0 && arr[i] <= arr[st.peek()]) {

st.pop();

}

if (st.size() == 0) {

lb[i] = -1;

} else {

lb[i] = st.peek();

}

st.push(i);

}

int max = 0;

for (int i = 0; i < arr.length; i++) {

int width = rb[i] - lb[i] - 1;

int area = width \* arr[i];

if (area > max) {

max = area;

}

}

System.out.println(max);

}

}

**5. Analysis**

Time Complexity: O(n)

Calculating NSE is O(n) we already know that. And to calculate the maximum area we are just doing a single loop. Hence that too results in O(n).

Space Complexity: O(n)

We are just using stacks for calculating the NSEs but for the maximum area we are not using any auxiliary space, so the space complexity will still be O(n).