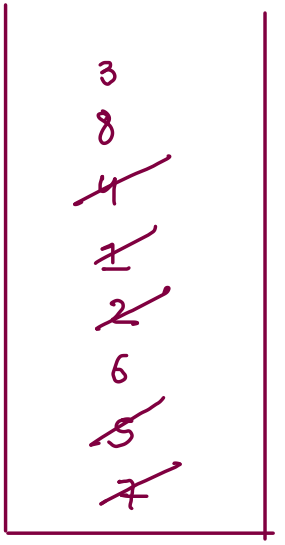
Next Greater Element To The Right

→ value based

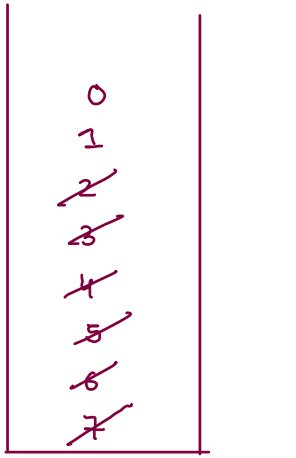
arr :	3	8	4	1	2	6	5	7
	0	1	2	3	4	5	6	7
i								
nge	8	-1	6	2	6	7	7	-1



Next Greater Element To The Right

→ index based

arr :	3	8	4	1	2	6	5	7
	0	1	2	3	4	5	6	7
nge	1	-1	5	4	5	7	7	-1



Stack → index

st.push(n-1) ; nge[n-1] = -1

```
for (int i=n-2; i>=0; i--) {
    while( st.size() && arr[st.peek()] <= arr[i]){
        st.pop();
    }
    if(st.size() == 0) {
        nge[i] = -1;
    }
    else {
        nge[i] = st.peek();
    }
    st.push(i);
}
```

503. Next Greater Element II

Given a circular integer array `nums` (i.e., the next element of `nums[nums.length - 1]` is `nums[0]`), return *the* ***next greater number*** *for every element in* `nums`.

The **next greater number** of a number `x` is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, return `-1` for this number.

arr : 3 (0) 8 (1) 4 (2) 1 (3) 2 (4)

8 -1 8 2 3

T : O(n)

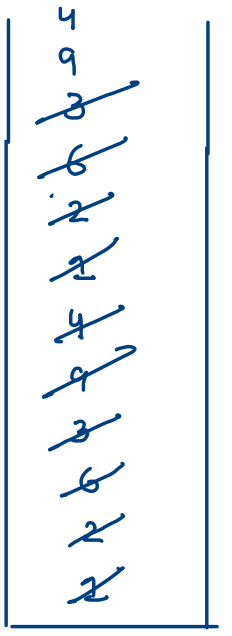
S : O(n)

arr :

4	9	3	6	2	1
0	1	2	3	4	5

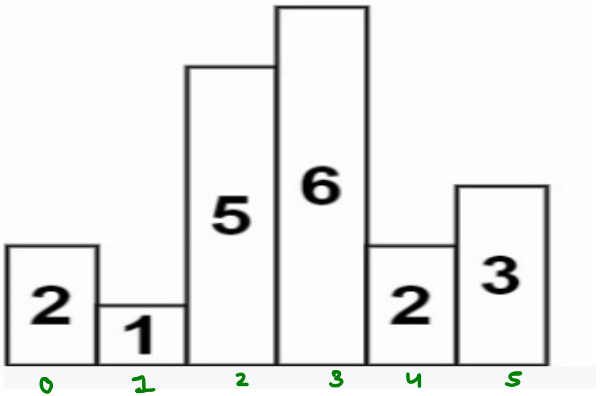
nge :

9	-1	6	9	4	4



84. Largest Rectangle in Histogram

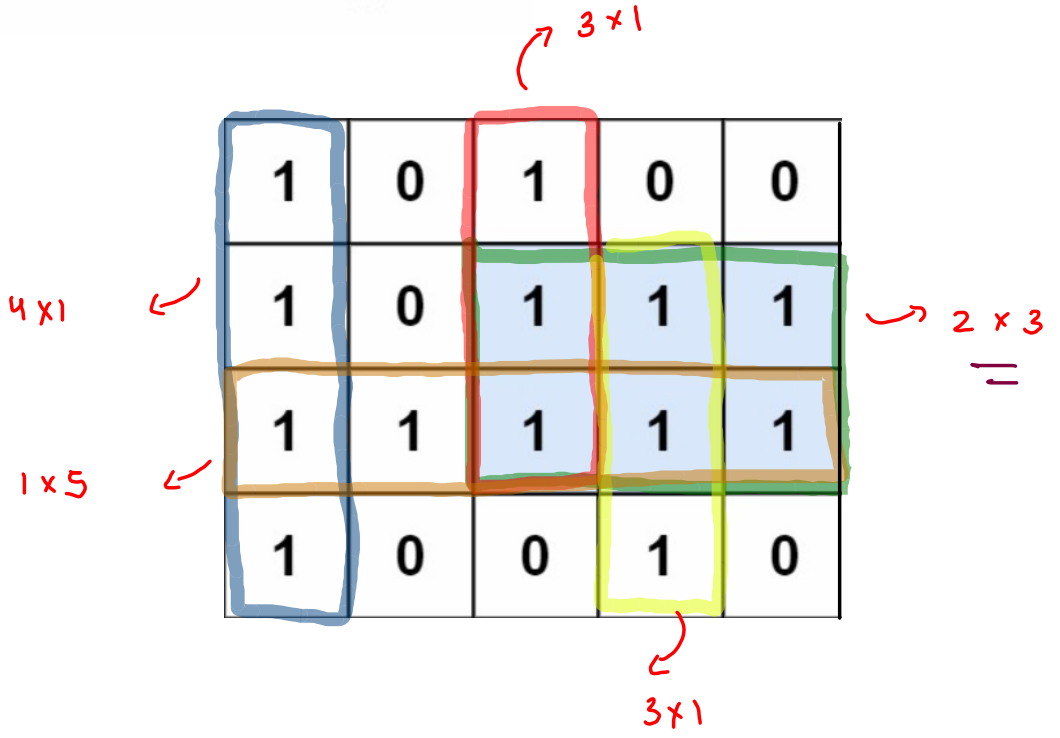
Input: heights = [2,1,5,6,2,3]
Output: 10
Explanation: The above is a histogram where width of each bar is 1.
The largest rectangle is shown in the red area, which has an area = 10 units.



$w = nsr[i] - nsl[i] - 1$

	0	1	2	3	4	5
nsl	-1	-1	1	2	1	4
nsr	1	6	4	4	6	6
w	1	6	2	1	4	1
ar	2	6	10	6	8	3

85. Maximal Rectangle



h x w

ans = 6

1	0	1	0	0
1	0	1	1	1
1	1	1	1	1
1	0	0	1	0

Jah

arr: 1 0 1 0 0 — 1

arr: 2 0 2 1 1 — 3

arr: 3 1 3 2 2 — 6

arr: 4 0 0 3 0 — 4

```java
public int maximalRectangle(char[][] matrix) {
    int n = matrix.length;
    int m = matrix[0].length;
    int[]arr = new int[m];
    int max = 0;

    for(int i=0; i < n;i++) {
        for(int j=0; j < m;j++) {
            if(matrix[i][j] == '0') {
                arr[j] = 0;
            }
            else {
                arr[j] += 1;
            }
        }

        int area = largestRectangleArea(arr);
        max = Math.max(area,max);
    }

    return max;
}
```

1	0	1	0	0
1	0	1	1	1
1	1	1	1	1
1	0	0	1	0

i

4 0 0 3 0

max = 6

946. Validate Stack Sequences

Given two integer arrays `pushed` and `popped` each with distinct values, return `true` *if this could have been the result of a sequence of push and pop operations on an initially empty stack, or* `false` *otherwise.*

pushed : 1 2 3 4 5

popped : 3 2 1 4 5

true

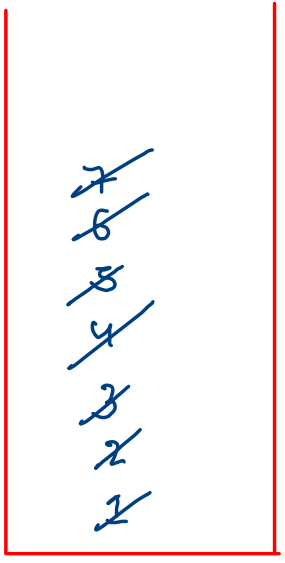
pushed : 1 2 3 4 5

popped : 4 3 1 5 2

false

push: 1 2 3 4 5 6 7

pop: 4 3 5 7 6 2 1



c = 1 + 1 + 1 + 1 + 1 + 1 + 1

```
while(j < n) {
    if(st.size() > 0 && popped[j] == st.peek()) {
        st.pop();
        j++;
        c++;
    }
    else if(i < n){
        st.push(pushed[i]);
        i++;
    }
    else {
        return false;
    }
}
```

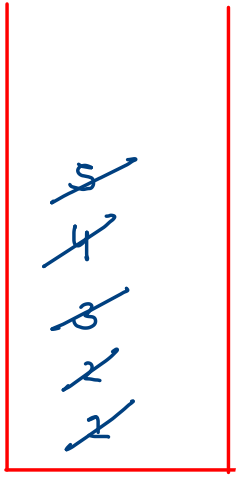
pushed : 1 2 3 4 5

i

popped : 3 2 1 4 5

j

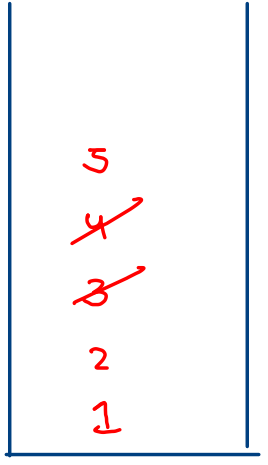
c = ~~1~~ 2 ~~3~~ ~~4~~ 5



```
while(j < n) {
    if(st.size() > 0 && popped[j] == st.peek()) {
        st.pop();
        j++;
        c++;
    }
    else if(i < n){
        st.push(pushed[i]);
        i++;
    }
    else {
        return false;
    }
}
```

pushed : 1 2 3 4 5

popped : 4 3 1 5 2



c = ~~1~~ 2