

## AGENDA

### Stack:

- Based on the concept of Last In First Out (LIFO).
- Real-life examples of stacks: simple
- Major operations in stacks:
  - **Push**: Add an element to the stack if memory is available
  - **Pop**: Remove the top element of the stack
  - **Peek or Top**: Return the top element of the stack
  - **size()**: returns the size of the stack
  - **isEmpty()**: checks if the stack is empty or not
- Can be implemented using linked list or arrays:
  - Arrays are preferred due to less dynamic involvement of memory allocation!
  - LinkedList is preferred when the maximum possible size is not known
  - Inbuilt stack functions/classes are safe to use!
- Problems:
  - Check if Parentheses are Balanced.
  - Next Greater Element.
  - Largest Rectangular Area in Histogram.
  - Max of Min for every Window Size.

## Stack Problems:

### Check If Parentheses are Balanced:

Given an expression string **x**. Examine whether the pairs and the orders of "{", "}", "(", ")", "[", "]" are correct in exp.

For example, the function should return 'true' for exp = "[()]{}{[()()]()}" and 'false' for exp = "[()]".

#### **Input:**

{([)])}

#### **Output:**

true

#### **Explanation:**

{ ( [ ] ) }. Same colored brackets can form balanced pairs, with 0 number of unbalanced bracket.

#### **Input:**

( [ ]

#### **Output:**

false

#### **Explanation:**

( [ ]. Here square bracket is balanced but the small bracket is not balanced and Hence , the output will be unbalanced.

## Next Greater Element

Given an array **arr[ ]** of size **N** having distinct elements, the task is to find the next greater element for each element of the array in order of their appearance in the array.

Next greater element of an element in the array is the nearest element on the right which is greater than the current element. If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

### **Input:**

N = 5, arr[] [6 8 0 1 3]

### **Output:**

8 -1 1 3 -1

### **Explanation:**

In the array, the next larger element to 6 is 8, for 8 there is no larger elements hence it is -1, for 0 it is 1, for 1 it is 3 and then for 3 there is no larger element on right and hence -1.

Ques-1: Can you now write an algorithm to find the immediate previous greater element for each element in the array?

Ques-2: Using the knowledge of finding immediate previous and next greater elements, can you write similar algorithms to find the immediate previous and next **smaller** elements for each element in the array?

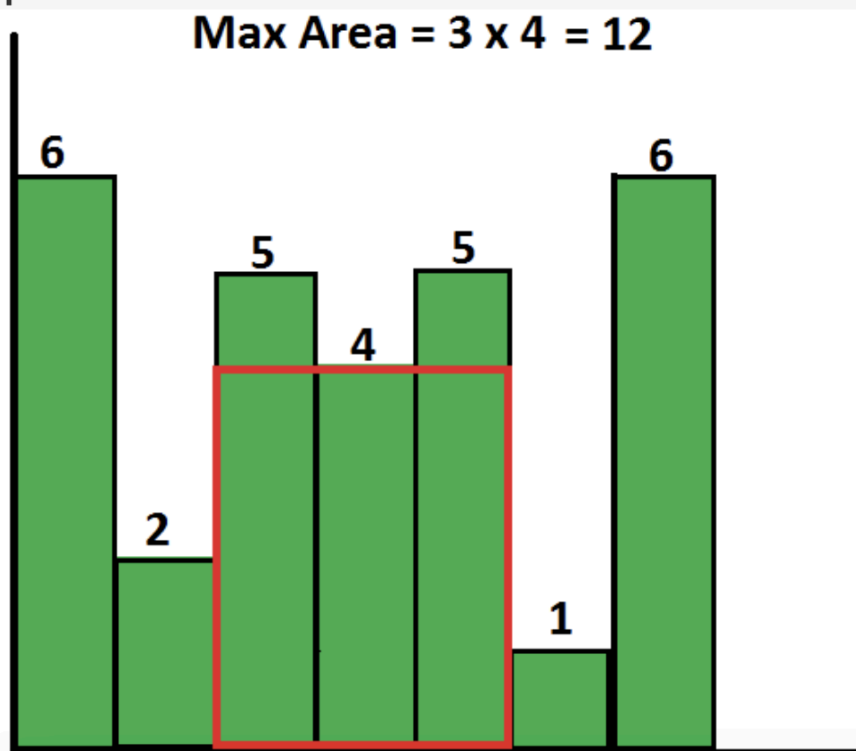
If the answer to both of the above questions is "YES", try solving this:

Find the largest rectangular area possible in a given histogram where the largest rectangle can be made of a number of contiguous bars. For simplicity, assume that all bars have the same width and the width is **1 unit**, there will be **N** bars height of each bar will be given by the array **arr**.

N = 7  
arr[] = {6,2,5,4,5,1,6}

**Output:** 12

**Explanation:**



## Max of Min for every Window Size

You are given an integer array **arr[]**, the task is to find the **maximum of minimum values** for every window size **k** where  **$1 \leq k \leq \text{arr.size}()$** .

For each window size **k**, consider all contiguous subarrays of length **k**, determine the minimum element in each subarray, and then take the maximum among all these minimums.

Return the results as an array, where the element at index **i** represents the answer for window size **i+1**.

### Examples :

**Input:** arr[] = [10, 20, 30, 50, 10, 70, 30]

**Output:** [70, 30, 20, 10, 10, 10, 10]

**Explanation:**

Window size 1: minimums are [10, 20, 30, 50, 10, 70, 30], maximum of minimums is 70.

Window size 2: minimums are [10, 20, 30, 10, 10, 30], maximum of minimums is 30.

Window size 3: minimums are [10, 20, 10, 10, 10], maximum of minimums is 20.

Window size 4–7: minimums are [10, 10, 10, 10], maximum of minimums is 10.

**Input:** arr[] = [10, 20, 30]

**Output:** [30, 20, 10]

**Explanation:**

Window size 1: minimums of [10], [20], [30], maximum of minimums is 30.

Window size 2: minimums of [10, 20], [20, 30], maximum of minimums is 20.

Window size 3: minimums of [10, 20, 30], maximum of minimums is 10.