**STACK**

Sameer Kushwaha

**Problem 1:** [**20. Valid Parentheses**](https://leetcode.com/problems/valid-parentheses/)

**Input:** s = "()"

**Output:** true

**Input:** s = "(] “

**Output:** false

**Solution:**



**Problem: Stock Span**

**Solution:**

**input:** 5 30 35 45 40 38 35 **outputs:** 1 2 3 1 1 END



**Problem: Next smaller element**

**Solution:**





**Problem: Next Greater Element (Medium)**

**Solution:**

**Sample Input**

2

4

11 13 21 3

5

11 9 13 21 3

**Sample Output**





**Problem**: Daily Temperature

**Solution:**

**Sample Input**

8

73 74 75 71 69 72 76 73

**Sample Output**

1 1 4 2 1 1 0 0



**Problem:** Given a stack, delete the **middle element**of the stack without using any additional data structure.  
**Middle element: -**floor((size\_of\_stack+1)/2) (1-based indexing) from bottom of the stack.

**Input**:

Stack = {10, 20, 30, 40, 50}

**Output**:

ModifiedStack = {10, 20, 40, 50}

**Explanation**:

If you print the stack the bottom-most element will be 10 and the top-most element will be 50. Middle element will be element at index 3 from bottom, which is 30. Deleting 30, stack will look like {10 20 40 50}.

**Solution:**



**Problem: Reverse A Stack**

**Input:**

St = {3,2,1,7,6}

**Output:**

{6,7,1,2,3}  
**Explanation:**  
Input stack after reversing will look like the stack in the output.

**Solution:**



**Problem:** [**155. Min Stack**](https://leetcode.com/problems/min-stack/)

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

* MinStack() initializes the stack object.
* void push(int val) pushes the element val onto the stack.
* void pop() removes the element on the top of the stack.
* int top() gets the top element of the stack.
* int getMin() retrieves the minimum element in the stack.

You must implement a solution with O(1) time complexity for each function.

**Example 1:**

**Input**

["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[]]

**Output**

[null,null,null,null,-3,null,0,-2]

**Explanation**

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); // return -3

minStack.pop();

minStack.top(); // return 0

minStack.getMin(); // return -2

**Solution:**



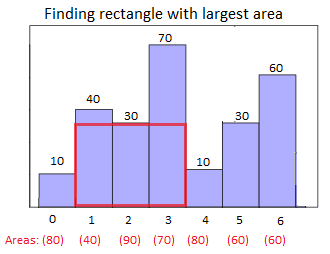
**Or...**



**Or..**



**Problem: HISTOGRAM**

Find the largest rectangular area possible in a given histogram where the largest rectangle can be made of a number of contiguous bars.  


**Input Format**

First line contains a single integer N, denoting the number of bars in th histogram.  
Next line contains N integers, ith of which, denotes the height of ith bar in the histogram.

**Constraints**

1<=N<=10^6  
Height of each bar in histogram <= 10^10

**Output Format**

Output a single integer denoting the area of the required rectangle.

**Sample Input**

5

1 2 3 4 5

**Sample Output**

9

**Explanation**

The largest rectangle can be obtained of breadth=3 and length=3. Its starting index is 2 and ending index is 4 and it has a height of 3. Hence area = 3\*3 = 9

**Solution:**



**Problem:** Redundant Brackets

**Solution:**

  
 }  
 return false;  
 }  


**Problem:**

**Solution:**

**Problem:**

**Solution:**

**Problem:**

**Solution:**

**Problem:**

**Solution:**

**Problem:**

**Solution:**

**Problem:**

**Solution:**

**Problem:**

**Solution:**

**Problem:**

**Solution:**

**Problem:**

**Solution:**

**Problem:**

**Solution:**

**Problem:**

**Solution:**

**Problem:**

**Solution:**

**Problem:**

**Solution:**

**Problem:**

**Solution:**

**Problem:**

**Solution:**