<https://leetcode.com/problems/min-stack/>

**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 = 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

Constraints:

-231 <= val <= 231 - 1

Methods pop, top and getMin operations will always be called on non-empty stacks.

At most 3 \* 104 calls will be made to push, pop, top, and getMin.

**Method 1: (Brute Force)**

Use two vectors or two stacks one for storing elements, other for storing minimum obtained at each position.

Time Complexity: O(1) *[for each function]*

Space Complexity: O(n) *[]*

class MinStack {

public:

    vector<int> s, minS;

    MinStack() {

    }

    void push(int val) {

        s.push\_back(val);

        if(s.size()==1 || val < minS.back())

            minS.push\_back(val);

        else minS.push\_back(minS.back());

    }

    void pop() {

        if(!s.empty()){

            s.pop\_back();

            minS.pop\_back();

        }

    }

    int top() {

        return s.back();

    }

    int getMin() {

        return minS.back();

    }

};