# Min Stack Solution (LeetCode 155) - Hinglish

## Logic Explanation:

Min Stack Problem - LeetCode 155

Logic Explanation:

------------------

Maine ek standard library stack data structure ka use kiya aur do stacks banaye:

1. 'st' - Main stack jo saare elements ko store karega.
2. 'gt' - Ek auxiliary stack jo har step pe minimum element ka track rakhega.

Push Operation:

- Jab ek value 'st' me push ho rahi hai, to 'gt' ke top ko check karna hai:
  - Agar 'st' empty hai, to dono stacks me wahi value push karni hai.
  - Agar new value 'gt' ke top se chhoti hai, to usko 'gt' me push karna hai.
  - Warna, 'gt' ka top element wapas 'gt' me push karna hai taaki current minimum maintain rahe.

Pop Operation:

- 'st' aur 'gt' dono se top element hata dena hai.

Top Operation:

- 'st.top()' return karna hai.

GetMin Operation:

- 'gt.top()' return karna hai jo minimum element hoga.

Is approach me **saare operations O(1) time complexity me complete hote hain**.

## Code Snippet:

```
// First solution for LeetCode problem 155
class MinStack {
public:
    stack<int> st;
    stack<int> gt;

    MinStack() {
        // Constructor (not needed in this method)
    }

    void push(int val) {
        if(st.size() == 0) {
            st.push(val);
            gt.push(val);
        } else {
            st.push(val);
            gt.push(min(val, gt.top()));
        }
    }

    void pop() {
        st.pop();
        gt.pop();
    }

    int top() {
        return st.top();
    }

    int getMin() {
        return gt.top();
    }
};
```

## Complexity Analysis:

Complexity Analysis:

--------------------

- Push Operation: O(1) - Har push operation constant time me complete hota hai.

- Pop Operation: O(1) - Dono stacks se ek element hatana constant time me hota hai.

- Top Operation: O(1) - Stack ka top access karna constant time ka kaam hai.

- GetMin Operation: O(1) - 'gt' ke top se minimum value lena constant time me hota hai.

Is approach ka **overall time complexity O(1) hai**, jo ise ek optimal solution banata hai.

## Test Cases aur Execution:

Test Cases aur Step-by-Step Execution:

--------------------------------------

Test Case 1:

------------

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

Input: [[],[-2],[0],[-3],[],[],[],[]]

Expected Output: [null,null,null,null,-3,null,0,-2]

Step-by-Step Execution:

1. push(-2) -> st = [-2], gt = [-2] (min -2 hai)
2. push(0)  -> st = [-2, 0], gt = [-2, -2] (min -2 hi rahega)
3. push(-3) -> st = [-2, 0, -3], gt = [-2, -2, -3] (new min -3)
4. getMin() -> -3 return karega (gt ka top)
5. pop()    -> -3 hata diya dono stacks se
6. top()    -> 0 return karega (st ka top)
7. getMin() -> -2 return karega (gt ka top)

Test Case 2:

------------

Operations: ["MinStack","push","push","push","pop","getMin"]

Input: [[],[5],[1],[6],[],[]]

Expected Output: [null,null,null,null,null,1]

Step-by-Step Execution:

1. push(5)  -> st = [5], gt = [5] (min 5)
2. push(1)  -> st = [5, 1], gt = [5, 1] (new min 1)
3. push(6)  -> st = [5, 1, 6], gt = [5, 1, 1] (min 1 hi rahega)
4. pop()    -> 6 remove kiya dono stacks se

5. getMin() -> 1 return karega (gt ka top)

Test Case 3:

------------

Operations: ["MinStack","push","push","push","pop","pop","getMin"]

Input: [[],[10],[20],[5],[],[],[]]

Expected Output: [null,null,null,null,null,null,10]

Step-by-Step Execution:

1. push(10) -> st = [10], gt = [10] (min 10)
2. push(20) -> st = [10, 20], gt = [10, 10] (min 10 hi rahega)
3. push(5)  -> st = [10, 20, 5], gt = [10, 10, 5] (new min 5)
4. pop()    -> 5 remove kiya dono stacks se
5. pop()    -> 20 remove kiya dono stacks se
6. getMin() -> 10 return karega (gt ka top)