Min Stack (LeetCode 155) - Hinglish Explanation

Problem Statement:

Ek stack design karna hai jo ye 4 operations O(1) time complexity me support kare:

- push(int val): Stack me ek naya element push kare.

- pop(): Stack ke top se ek element hataye.

- top(): Stack ka top element return kare.

- getMin(): Stack me jo bhi minimum element hai usko return kare.

Optimized Approach (Mathematical Trick):

Ek extra stack use karne ki jagah hum ek single stack ke andar hi min value maintain kar sakte hain.

Data Members:

1. stack<long long> st; (Elements store karne ke liye)

2. long long min; (Current minimum element store karne ke liye)

Logic:

- Hum ek 'min' variable rakhenge jo hamesha stack ka current minimum store karega.

- Agar naya element min se bada hai, toh seedha push kar denge.

- Agar naya element min se chhota ya equal hai:

** - Hum actual value ki jagah '2*x - min' push karenge jo pichla min encode karega.**

- min ko update kar denge.

Pop operation me:

- Agar 'st.top() < min' hai, toh iska matlab encoded value hai, toh hum pichla min 'min = 2*min - st.top()' se recover kar lenge.

Code Implementation:

```
class MinStack {
  public:
      stack<long long> st;
      long long min;
      MinStack() {
          min = LLONG_MAX;
      }

      void push(int val) {
         long long x = (long long)val;
         if(st.size()==0){
          st.push(x);
          min = x;
         }
         else if(x>min){
          st.push(x);
         }
         else{
          st.push(2*x-min);
          min = x;
         }
      }

      void pop() {
         if(st.top()<min){
          min = 2*min - st.top();
         }
         st.pop();
      }

      int top() {
          if(st.top()<min) return (int)min;
          else return (int)st.top();
```

```
    }

    int getMin() {
        return (int)min;
    }
};
```

Test Cases:

Test Case 1:

Input:

MinStack obj;

obj.push(3);

obj.push(5);

cout << obj.getMin() << endl; // Output: 3

obj.push(2);

obj.push(1);

cout << obj.getMin() << endl; // Output: 1

obj.pop();

cout << obj.getMin() << endl; // Output: 2

Test Case 2:

Input:

MinStack obj;

obj.push(10);

obj.push(20);

obj.push(5);

obj.push(30);

```
cout << obj.getMin() << endl;  // Output: 5
obj.pop();
cout << obj.getMin() << endl;  // Output: 5
obj.pop();
cout << obj.getMin() << endl;  // Output: 10
```

Time aur Space Complexity Analysis:

- push(): O(1) (Direct stack operation)
- pop(): O(1) (Direct stack operation)
- top(): O(1) (Stack ka top access karna)
- getMin(): O(1) (Min value directly return karna)

Space Complexity: O(N) (Worst case me sabhi elements ko stack me encoded store karna pad sakta hai)