Implement Min Stack : O(2N) and O(N) Space Complexity
Problem Statement: Implement Min Stack | O(2N) and O(N) Space Complexity. Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.
Examples:

Input Format:["MinStack", "push", "push", "push", "getMin", "pop", "top", "getMin"]
[
[ ], [-2], [0], [-3], [ ], [ ], [ ], [ ]
]

Result: [null, null, null, null, -3, null, 0, -2]
Explanation:
stack < long long > st
st.push(-2); Push element in stack
st.push(0); Push element in stack
st.push(-3); Push element in stack
st.getMin(); Get minimum element fromstack
st.pop(); Pop the topmost element
st.top(); Top element is 0
st.getMin(); Minimum element is -2

Solution:
Disclaimer: *Don't jump directly to the solution, try it out yourself first.*

Solution 1: Using pairs to store the value and minimum element till now.
Approach: The first element in the pair will store the value and the second element will store the minimum element till now.
When the first push operation comes in we will push the value and store it as minimum itself in the pair.
In the second push operation, we will check if the top element's minimum is less than the new value. If it is then we will push the value with minimum as the previous top's minimum. To get the getMin element to take the top's second element.
Code:

- C++ Code
- Java Code

```
class MinStack {
  stack < pair < int, int >> st;

  public:
    void push(int x) {
      int min;
      if (st.empty()) {
        min = x;
      } else {
        min = std::min(st.top().second, x);
      }
      st.push({x,min});
    }

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

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

  int getMin() {
    return st.top().second;
  }
};
```

Time Complexity: O(1)

Space Complexity: O(2N)

Solution 2:
Approach:
Let's take a variable that stores the minimum number. So whenever a push operation comes in just take that number put it in the stack and update the variable to the number.
Push operation:
Now if there is a push operation just check whether that number is less than the min number. If it is smaller than min we will push a modified value which is a push(2 * Val – min) into the stack and will update min to the value of the original number. If it's not then we will just push it as it is.

getMin() operation:
We will just return the value of min.
Top operation:
While returning the top value we know that it is a modified value. We will check if the top value is lesser than min, If it is then we will return the min as the top value.
Pop operation:
While making pop we will check if the top value is lesser than min, If it is then we must update our min to its previous value. In order to do that min = (2 * min) – (modified value) and we will pop the element.
Code:

- C++ Code
- Java Code

```
class MinStack {
    stack < long long > st;
    long long mini;
    public:
        /** initialize your data structure here. */
        MinStack() {
            while (st.empty() == false) st.pop();
            mini = INT_MAX;
        }

    void push(int value) {
        long long val = Long.valuevalue;
        if (st.empty()) {
            mini = val;
            st.push(val);
        } else {
            if (val < mini) {
                st.push(2 *val*1LL - mini);
                mini = val;
            } else {
                st.push(val);
            }
        }
    }

    void pop() {
        if (st.empty()) return;
        long long el = st.top();
        st.pop();

        if (el < mini) {
            mini = 2 * mini - el;
        }
    }

    int top() {
        if (st.empty()) return -1;

        long long el = st.top();
        if (el < mini) return mini;
        return el;
    }

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

Time Complexity: O(1)

Space Complexity: O(N)