[Click on the logo to find **Problem Statement**]

Intuition

The `MinStack` class is designed to maintain a stack of elements while efficiently keeping track of the minimum element at any given point. The class utilizes two lists, `stack` and `min`, where `stack` holds the elements in the order they are pushed, and `min` keeps track of the minimum element encountered so far.

When a new element is pushed onto the stack, it is added to both `stack` and `min`. The crucial aspect is that for each new element, the minimum is updated only if the new element is smaller than the current minimum. This ensures that the `min` list always reflects the minimum element at the corresponding position in the `stack`.

Popping an element involves removing the top element from both `stack` and `min`, maintaining the consistency of the minimum element tracking. The `top` method retrieves the last element pushed onto the stack, and the `getMin` method retrieves the last element from the `min` list, representing the current minimum.

The efficiency of this design lies in constant time complexity for push, pop, top, and getMin operations, making it suitable for scenarios where constant-time performance is crucial, especially when dealing with large datasets.

Approach

1. **Initialization:**
    - Create two empty lists, `stack` and `min`, to represent the stack and keep track of the minimum element.
2. **Push Operation:**
    - When a new element is pushed onto the stack:
        - Append the element to the `stack` list.
        - Check if the `min` list is empty or if the new element is smaller than the current minimum (the last element in the `min` list).
        - If the above condition is true, append the new element to the `min` list; otherwise, append the current minimum to maintain consistency.

3. **Pop Operation:**
    - When an element is popped from the stack:
        - Remove the last element from both the `stack` and `min` lists, ensuring synchronization between the two.
4. **Top Operation:**
    - Retrieve the last element from the `stack` list, representing the element at the top of the stack.
5. **GetMin Operation:**
    - Retrieve the last element from the `min` list, representing the current minimum element in the stack.

Complexity

- Time complexity: O(1)
- Space complexity: O(1)

Code

```
class MinStack:

    def __init__(self):
        self.min = []
        self.stack = []

    def push(self, x):
        self.stack.append(x)
        if not self.min or x < self.min[-1]:
            self.min.append(x)
        else:
            self.min.append(self.min[-1])

    def pop(self):
        self.stack.pop()
        self.min.pop()

    def top(self):
        return self.stack[-1]

    def getMin(self):
        return self.min[-1]
```

```
# Your MinStack object will be instantiated and called as such:
# obj = MinStack()
# obj.push(val)
# obj.pop()
# param_3 = obj.top()
# param_4 = obj.getMin()
```

If you want to see more solutions to coding problems, you can visit: