Design a Special Stack to Get Maximum value in O(1) Time Complexity

You are tasked with designing a stack data structure that supports efficient push, pop, getTop, and getMax operations. These operations should be performed in constant time complexity (O(1)), while also ensuring that space usage is efficient

Approach 1: MaxStack class using two stacks to track maximum value

In this approach, two separate stacks are used to maintain the elements of the main stack and the corresponding maximum values at each step. When an element is pushed onto the stack, the maximum value is updated in the auxiliary maximum stack.

Functions:

- **Push Operation:**
    - Description: Pushes the given value onto the main stack and updates the maximum value stack.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(N)**
- **Pop Operation:**
    - Description: Pops the top element from both the main stack and the maximum value stack.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(N)**
- **GetTop Operation:**
    - Description: Returns the top element of the main stack without removing it.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
- **GetMax Operation:**
    - Description: Returns the maximum value present in the maximum value stack.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**

Approach 2: MaxStackDifference class using a single stack and difference to track maximum value

In this approach, a single stack is used to store the elements, along with a variable to track the current maximum value. When pushing an element, the difference between the element and the current maximum is stored if the element is larger than the current maximum.

Functions:

- **Push Operation:**
    - Description: Pushes the given value onto the stack while updating the current maximum value.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(N)**
- **Pop Operation:**
    - Description: Pops the top element from the stack and updates the current maximum value if needed.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(N)**
- **GetTop Operation:**
    - Description: Returns the top element of the stack without removing it.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
- **GetMax Operation:**
    - Description: Returns the current maximum value stored in the variable.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**

Approach 3: SpecialStackPair class using pairs to track both the element and the maximum value

In this approach, a single stack is used to store pairs of elements and their corresponding maximum values. When pushing an element, the maximum value is updated and stored along with the element as a pair.

Functions:

- **Push Operation:**

    - Description: Pushes the given value onto the stack along with the corresponding maximum value.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(N)**
- **Pop Operation:**
    - Description: Pops the top element from the stack and updates the current maximum value accordingly.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(N)**
- **GetTop Operation:**
    - Description: Returns the top element of the stack without removing it.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
- **GetMax Operation:**
    - Description: Returns the maximum value stored in the top pair of the stack.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**

The Best Approach:

- **If space efficiency is important:** Choose Approach 2 (MaxStackDifference) as it uses a single stack and has a relatively efficient use of memory.
- **If ease of implementation is key:** Go with Approach 1 (MaxStack using Two Stacks) or Approach 3 (SpecialStackPair), as they have more straightforward implementations.
- **If you want a balance:** Consider Approach 3 (SpecialStackPair), which offers a good compromise between memory efficiency and ease of implementation.