Reverse Stack Elements CodeStudio

The program reverses the stack elements

Example: Consider the initial stack: 6 5 4 3 2 1

Output: 1 2 3 4 5 6

Approach 1: Reverses the stack using recursion

In this approach, the **reverseStack** function uses recursion to reverse the elements of the stack. The main idea is to pop the top element from the stack, then recursively reverse the remaining elements, and finally push the popped element at the bottom of the reversed portion.

Steps:

1. Pop the top element from the stack and store it temporarily.
2. Recursively call the **reverseStack** function to reverse the remaining elements.
3. Push the stored element at the bottom of the reversed portion of the stack.

Time Complexity:

- The function iterates through each element in the stack once.
- Since each element requires constant work (popping, pushing), the time complexity is linear.
- **Time Complexity**: **O(n), where n is the number of elements in the stack.**

Space Complexity:

- The space complexity of the recursive approach is influenced by the maximum depth of the call stack due to recursive calls.
- In the worst case, the depth of the call stack can be **n** (number of elements in the stack).
- **Space Complexity**: **O(n), due to the call stack space required for recursion.**

Approach 2: Reverses the stack using an auxiliary stack

In this approach, an auxiliary stack (**tempStack**) is used to temporarily store the elements while rearranging the original stack. The idea is to transfer elements from the input stack to the auxiliary stack, and then transfer them back from the auxiliary stack to the input stack in reversed order.

Steps:

1. Transfer elements from the input stack to the auxiliary stack.
2. Transfer elements back from the auxiliary stack to the input stack.

Time Complexity:

- The approach involves two iterations: one for transferring elements to the auxiliary stack and another for transferring them back.
- Both iterations are linear in nature.
- **Time Complexity**: **O(n), where n is the number of elements in the stack.**

Space Complexity:

- An auxiliary stack is used to store the elements temporarily.
- In the worst case, the auxiliary stack holds all the elements.
- **Space Complexity**: **O(n), due to the auxiliary stack space requirement.**

Approach 3: Reverses the stack using a queue

In this approach, a queue is used as a temporary container to reverse the stack. The elements from the input stack are transferred to the queue and then transferred back from the queue to the input stack.

Steps:

1. Transfer elements from the input stack to the queue.
2. Transfer elements back from the queue to the input stack.

Time Complexity:

- Similar to the auxiliary stack approach, this approach involves two linear iterations.
- **Time Complexity**: **O(n), where n is the number of elements in the stack.**

Space Complexity:

- A queue is used as a temporary container.
- In the worst case, the queue holds all the elements from the stack.
- **Space Complexity**: **O(n), due to the queue space requirement.**