Reverse Array after given Index Position CodeStudio

The given code aims to reverse a portion of an array. It provides two functions: **reverseArrayIterative** and **reverseArrayRecursive**, which reverse the array iteratively and recursively, respectively. The main function demonstrates the usage of these functions by reversing a portion of the array and printing the results.

Example: Let's consider the array **arr = {1, 4, 4, 5, 6, 7, 3}** and the target index **targetIndex = 3**. The code will reverse the elements in **arr** starting from **targetIndex** using both iterative and recursive approaches.

The array before reverse: 1 4 4 5 6 7 3

Applying Iterative reverse after index 3: 1 4 4 3 7 6 5

The array before reverse: 1 4 4 3 7 6 5

The array after recursive reverse after index 3: 1 4 4 3 5 6 7

Approach 1: Reverses the array iteratively

reverseArrayIterative: This function takes a vector **arr** and a target index **targetIndex**. It uses a two-pointer approach to reverse the portion of the array after the **targetIndex**. It initializes **start** as **targetIndex + 1** and **end** as **arr.size() - 1**. Then, it iterates until **start** is less than or equal to **end**, swapping the elements at the corresponding indices and incrementing **start** and decrementing **end**. This process effectively reverses the desired portion of the array.

For the reverseArrayIterative function, the time complexity is O(N), where N is the size of the array. It performs a linear scan over half of the array.

The space complexity of the function is O(1) because it don't use any extra space that grows with the input size.

Approach 2: Reverses the array recursively

reverseArrayRecursive: This function takes a vector **arr**, a starting index **start**, and an ending index **end**. It uses recursion to reverse the portion of the array between **start** and **end**. The base case checks if **start >= end**. If true, it means the entire portion has been reversed, and the function returns. Otherwise, it swaps the elements at **start** and **end** indices and recursively calls itself with **start+1** and **end-1** to reverse the remaining portion.

For the reverseArrayRecursive function, the time complexity is also O(N) as it performs a recursive call for each pair of elements to be swapped.

The space complexity of the function is O(1) because it don't use any extra space that grows with the input size.