Merge Two Sorted Arrays LeetCode

You are given two sorted arrays, **arr1** and **arr2**, of sizes **m** and **n** respectively. Your task is to merge the elements of **arr2** into **arr1** such that the merged array is also sorted.

Input:

arr1 = [1, 1, 3, 4, 9, 10, 12], arr2 = [2, 4, 9, 12]

Output: arr1 = [1, 1, 2, 3, 4, 4, 9, 9, 10, 12, 12]

Approach 1: Function to merge two sorted arrays using a temp array

The first approach uses a temporary array **temp** to merge the two arrays. It compares the elements from both arrays and stores the smaller one in **temp**. After merging, the contents of **temp** are copied back to **arr1**.

This approach has a time complexity of O(m+n).

This approach has a space complexity of O(m+n).

Approach 2: Function to merge two sorted arrays in-place.

The second approach is an optimized version that reduces the space complexity to O(1). It performs the merging operation directly in **arr1**, starting from the end of both arrays. It uses three pointers, **i**, **j**, and **k**, to iterate over **arr1**, **arr2**, and the merged result respectively. The elements are compared, and the larger one is placed at the end of **arr1**. If there are any remaining elements in **arr2**, they are copied to the beginning of **arr1**.

This approach also has a time complexity of O(m+n).

It uses only a constant amount of extra space for the three pointers, resulting in a space complexity of O(1).

Which approach to use:

- If you have enough memory space and don't have any constraints, Approach 1 can be used. It is easier to understand and modify the original array.
- If you want to optimize the space usage and don't want to create a new temporary array, Approach 2 is preferred. It modifies the original array in-place and has a lower space complexity.