Problem Recap

You need to merge two sorted arrays, nums1 and nums2, into a single sorted array stored inside nums1. The length of nums1 is m + n, where the first m elements represent the elements that should be merged, and the remaining are zeros to accommodate the elements of nums2.

Approach 1: Brute Force

Idea:

- Copy all elements of nums2 into nums1 starting after the first m elements.
- Then, sort the entire nums1 array.

```cpp
void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
    for (int i = 0; i < n; i++) {
        nums1[m + i] = nums2[i];
    }
    sort(nums1.begin(), nums1.end());
}
```

Complexity:

- **Time Complexity:** O((m + n) log(m + n)) due to the sorting step.
- **Space Complexity:** O(1) as sorting is done in place.

Approach 2: Better Approach

Idea:

- Use an auxiliary array to store the merged result and then copy it back to nums1.

```cpp
void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
    vector<int> temp(m + n);
    int i = 0, j = 0, k = 0;

    while (i < m && j < n) {
        if (nums1[i] < nums2[j]) {
            temp[k++] = nums1[i++];
        } else {
            temp[k++] = nums2[j++];
        }
    }

    while (i < m) {
        temp[k++] = nums1[i++];
    }

    while (j < n) {
        temp[k++] = nums2[j++];
    }

    for (int i = 0; i < m + n; i++) {
        nums1[i] = temp[i];
    }
}
```

Complexity:

- **Time Complexity:** O(m + n) because we are iterating through both arrays once.
- **Space Complexity:** O(m + n) due to the auxiliary array.

Approach 3: Optimal Approach (In-Place Merge)

Idea:

- Start merging from the end of nums1 and nums2.
- Place the largest elements at the end of nums1, moving backward.

```cpp
void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
    int i = m - 1;  // Index for the last element in nums1
    int j = n - 1;  // Index for the last element in nums2
    int k = m + n - 1;  // Index for the last position in nums1

    while (i >= 0 && j >= 0) {
        if (nums1[i] > nums2[j]) {
            nums1[k--] = nums1[i--];
        } else {
            nums1[k--] = nums2[j--];
        }
    }

    while (j >= 0) {
        nums1[k--] = nums2[j--];
    }
}
```

Complexity:

- **Time Complexity:** O(m + n) because we are iterating through both arrays from the end.
- **Space Complexity:** O(1) as no extra space is used.

Explanation of the Optimal Approach:

- **Step 1:** Start from the end of both nums1 and nums2. Compare the elements from the back.
- **Step 2:** Place the larger element at the end of nums1, moving backward.
- **Step 3:** If any elements remain in nums2 (because all elements in nums1 are already in place), copy them into nums1.

This in-place approach ensures we don't need any extra space and works efficiently in linear time.