Documentation for Merge Sorted Array

Problem Description

You are given two integer arrays nums1 and nums2, both sorted in non-decreasing order. You are also given two integers m and n, representing the number of elements in nums1 and nums2, respectively. The goal is to merge nums1 and nums2 into a single array sorted in non-decreasing order. The final sorted array should be stored inside the array nums1.

- nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored.
- nums2 has a length of n.

Example 1

Input:

nums1 = [1, 2, 3, 0, 0, 0]

m = 3

nums2 = [2, 5, 6]

n = 3

Output: [1, 2, 2, 3, 5, 6]

Explanation: The arrays we are merging are [1, 2, 3] and [2, 5, 6]. The result of the merge is [1, 2, 2, 3, 5, 6].

Example 2

Input:

nums1 = [1]

m = 1

nums2 = []

n = 0

Output: [1]

Explanation: The arrays we are merging are [1] and []. The result of the merge is [1].

Example 3

Input:

nums1 = [0]

m = 0

nums2 = [1]

n = 1

Output: [1]

Explanation: The arrays we are merging are [] and [1]. The result of the merge is [1].

Constraints:

- nums1.length == m + n
- nums2.length == n
- 0 <= m, n <= 200
- 1 <= m + n <= 200
- -10^9 <= nums1[i], nums2[j] <= 10^9

Follow-up:

Can you come up with an algorithm that runs in O(m + n) time?

Explanation

1. **Pointers Initialization:**

- p1 is initialized to the last index of the non-zero elements in nums1 (i.e., m 1).
- p2 is initialized to the last index of nums2 (i.e., n 1).
- p is initialized to the last index of nums1 (i.e., m + n 1).

2. **Merging in Reverse Order:**

- We compare elements from the end of nums1 and nums2.
- We place the larger element at the current end position (p) of nums1.
- We move the respective pointers (p1, p2, p) accordingly.

3. **Copy Remaining Elements:**

- If any elements remain in nums2 after the initial merge, we copy them into nums1.
- This is necessary because nums1 might have leftover zeros if nums2 had larger elements.

Time Complexity

- The algorithm runs in O(m + n) time because each element from both arrays is processed exactly once.

Space Complexity

- The algorithm uses O(1) extra space as it modifies nums1 in place without using additional arrays.

This approach efficiently merges the two sorted arrays within the given constraints and ensures that the final result is stored in nums1.