88. Merge Sorted Array

Easy | Topics | Companies | Hint

You are given two integer arrays `nums1` and `nums2`, sorted in **non-decreasing order**, and two integers `m` and `n`, representing the number of elements in `nums1` and `nums2` respectively.

Merge `nums1` and `nums2` into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array* `nums1`. To accommodate this, `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] with the underlined elements coming from nums1.

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].
Note that because m = 0, there are no elements in nums1.
The 0 is only there to ensure the merge result can fit
in nums1.
```

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?

Python:

```
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        """
        Do not return anything, modify nums1 in-place instead.
        """
```

```
        # Pointers for nums1, nums2, and the end position
        i, j, k = m - 1, n - 1, m + n - 1

        # Merge in reverse order
        while i >= 0 and j >= 0:
            if nums1[i] > nums2[j]:
                nums1[k] = nums1[i]
                i -= 1
            else:
                nums1[k] = nums2[j]
                j -= 1
            k -= 1

        # If any elements remain in nums2, copy them
        while j >= 0:
            nums1[k] = nums2[j]
            j -= 1
            k -= 1
```

JavaScript:

```
/**
 * @param {number[]} nums1
 * @param {number} m
 * @param {number[]} nums2
 * @param {number} n
 * @return {void} Do not return anything, modify nums1 in-place instead.
 */
var merge = function(nums1, m, nums2, n) {
    // Pointers for nums1, nums2, and the merged array
    let i = m - 1;       // Last element in nums1's valid part
    let j = n - 1;       // Last element in nums2
    let k = m + n - 1;   // Last index of nums1 (total length)

    // Merge in reverse order
    while (i >= 0 && j >= 0) {
        if (nums1[i] > nums2[j]) {
            nums1[k] = nums1[i];
            i--;
        } else {
            nums1[k] = nums2[j];
            j--;
        }
        k--;
    }
```

```
    // If nums2 still has remaining elements, copy them
    while (j >= 0) {
        nums1[k] = nums2[j];
        j--;
        k--;
    }
};
```

Java:

```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        // Pointers for nums1, nums2, and the end of nums1
        int i = m - 1;       // Last index of nums1's initial elements
        int j = n - 1;       // Last index of nums2
        int k = m + n - 1;   // Last index of nums1's total capacity

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

        // If nums2 still has elements, copy them
        while (j >= 0) {
            nums1[k] = nums2[j];
            j--;
            k--;
        }
    }
}
```