Merge Sorted Array

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`.

Examples

```
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.
```

Approach 1: Brute Force – Simple Concatenate + Sort

Overwrite the trailing zeros in `nums1` (from index `m` onwards) with all elements of `nums2`. Sort the whole `nums1` array.

Dry Run:

```
Input:
nums1 = [1, 2, 3, 0, 0, 0], m = 3
nums2 = [2, 5, 6], n = 3

Step 1: Copy nums2 into nums1
i = 3 to 5:
nums1[3] = nums2[0] = 2
nums1[4] = nums2[1] = 5
nums1[5] = nums2[2] = 6
=> nums1 = [1, 2, 3, 2, 5, 6]

Step 2: Sort the array
nums1.sort((a, b) => a - b)
=> [1, 2, 2, 3, 5, 6]
```

Time and Space Complexity

Time: O((m+n) log(m+n)) due to sorting.
Space: O(1) extra (in-place).

JavaScript | C++ | C | Java | Python

```javascript
var merge = function(nums1, m, nums2, n) {
  for (let i = m; i < nums1.length; i++) {
    nums1[i] = nums2[i - m];
  }
  nums1.sort((a, b) => a - b);
};
```

Approach 2: Two-Pointer Method

Instead of sorting at the end, this algorithm merges the arrays in sorted order using two pointers:

Copy the first m elements of nums1 into a temporary array (nums1Copy).
Use two pointers p1 (for nums1Copy) and p2 (for nums2) to compare elements.
At each index i of nums1, place the smaller of the elements from nums1Copy[p1] and nums2[p2].
Repeat until nums1 is fully filled with the merged sorted elements.

Dry Run

```
Input:
nums1 = [1,2,3,0,0,0], m = 3
nums2 = [2,5,6], n = 3

Execution:
nums1Copy = [1,2,3], p1 = 0, p2 = 0

nums1Copy[0]=1 < nums2[0]=2 → nums1[0] = 1, p1++
nums1Copy[1]=2 == nums2[0]=2 → nums1[1] = 2, p2++
nums1Copy[1]=2 < nums2[1]=5 → nums1[2] = 2, p1++
nums1Copy[2]=3 < nums2[1]=5 → nums1[3] = 3, p1++
p1==3 → only nums2 left
nums1[4] = 5, nums1[5] = 6

Final Output: nums1 = [1,2,2,3,5,6]
```

Time Complexity:

Copying the first m elements to nums1Copy takes O(m).
Merging the two sorted arrays takes O(m + n) because each index in nums1 is visited exactly once.
Overall: O(m + n)

Space Complexity:

You create a copy of the first m elements of nums1 in nums1Copy, which takes O(m) additional space.
Overall: O(m)

JavaScript | C++ | C | Java | Python

```javascript
var merge = function(nums1, m, nums2, n) {
    let nums1Copy = nums1.slice(0, m)
    let p1 = 0;
    let p2 = 0;
    for (let i = 0; i < m + n; i++) {
        if (p2 >= n || (p1 < m && nums1Copy[p1] < nums2[p2])) {
            nums1[i] = nums1Copy[p1];
            p1++;
        } else {
            nums1[i] = nums2[p2];
            p2++;
        }
    }
};
```

Optimal Approach

We have two sorted arrays:

nums1 with length m + n where the first m elements are valid.
nums2 with n elements.

The goal: merge nums2 into nums1 in sorted order in-place.

Start filling nums1 from the end (index m + n - 1), comparing the last elements of both arrays (nums1[m-1] and nums2[n-1]).

Place the larger element at the current last position.

Move pointers accordingly:

Decrement the pointer in nums1 or nums2.
Decrement the position pointer for placement.

If nums2 is exhausted first, merging is done.

If nums1 is exhausted first, copy remaining elements of nums2.

Time Complexity (TC)

We traverse the combined length of both arrays exactly once, i.e., m + n times.
Each step is O(1).
Overall: O(m + n)

Space Complexity (SC)

No extra significant space is used (in-place).
Only a few variables (p1, p2, i) are used.
Overall: O(1) (constant space)

JavaScript | C++ | C | Java | Python

```javascript
var merge = function(nums1, m, nums2, n) {
  let p1 = m - 1;
  let p2 = n - 1;

  for (let i = m + n - 1; i >= 0; i--) {
    if (p2 < 0) break;

    if (p1 >= 0 && nums1[p1] > nums2[p2]) {
      nums1[i] = nums1[p1--];
    } else {
      nums1[i] = nums2[p2--];
    }
  }
};
```