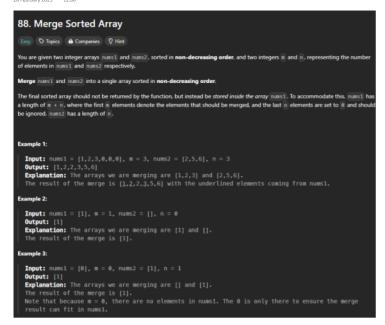
28 February 2025 12:00





Code:

```
def merge(nums1, m, nums2, n):
    i, j, k = m - 1, n - 1, m + n - 1 # Pointers for nums1, nums2, and merge position

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 there are remaining elements in nums2, copy them over
while j >= 0:
    nums1[k] = nums2[j]
    j -= 1
    k -= 1
```

Time Complexity:

• $O(m + n) \rightarrow We$ traverse both arrays once.

Space Complexity:

O(1) → We modify nums1 in-place without extra space.

Example Walkthrough:

Input:

Understanding the Problem

You are given two sorted integer arrays:

- 1. nums1: A list that is already sorted and has extra space at the end (filled with 0s).
- 2. nums2: A smaller sorted list that needs to be merged into nums1.

You are also given two numbers:

- m: The number of actual elements in nums1 (ignoring trailing 0s).
- n: The number of elements in nums2.

Your task is to **merge nums2 into nums1** so that nums1 becomes a single sorted array.



Constraints and Hints

- The final array should not be returned; instead, modify nums1 in-place.
- nums1 has enough space to hold all elements (length m + n).
- Since nums1 and nums2 are already sorted, an efficient approach would be to use a two-pointer technique to merge them from the end.

Simplified Thought Process

- 1. Start from the end of both arrays.
- 2. Compare the last elements of nums1 and nums2.
- 3. Place the larger one at the last available position in nums1.
- **4. Move backward** and repeat the process until all elements are merged.
- If elements remain in nums2, copy them over (since nums1 elements are already in place).

Explanation:

Move pointers accordingly (decrease i, j, and k).

```
python

② Copy № Edit

i, j, k = m - 1, n - 1, m + n - 1

• i = m - 1 → Points to the last valid element in nums1.

• j = n - 1 → Points to the last element in nums2.

• k = m + n - 1 → Points to the last position in nums1 (where we place elements).
```

```
Step 2: Compare and Merge Elements

python

### Copy 19 Edit

while i >= 0 and j >= 0:
    if nums1[i] > nums2[j]:
        nums1[k] = nums1[i] # Place the larger element at nums1[k]
        i -= 1 # Move left in nums1

else:
        nums1[k] = nums2[j] # Place the nums2 element at nums1[k]
        j -= 1 # Move left in nums2
        k -= 1 # Move left in merged array

* We compare the last elements of nums1 and nums2.

• Place the larger one at the last available position (nums1[k]).
```

Example Walkthrough:

```
python

nums1 = [1, 2, 3, 0, 0, 0] # m = 3 (actual elements: [1, 2, 3])
m = 3
nums2 = [2, 5, 6] # n = 3
n = 3
```

New Section 1 Page 2

Initial Setup: We have three pointers: • i = m - 1 = 2 (points to the last valid element in nums1) • j = n - 1 = 2 (points to the last element in nums2) • k = m + n - 1 = 5 (points to the last index of nums1) Step-by-step Merging: Step nums1 (current state) Action Pointers (i, j, k) [1, 2, 3, 0, 0, 0] Compare nums1[2] = 3 and nums2[2] = 6 i = 2, j = 2, k = 5Step 1 [1, 2, 3, 0, 0, 6] 6 > 3, place 6 at nums1[5] j - 1, k - 4 Step 2 [1, 2, 3, 0, 5, 6] 5 > 3, place 5 at nums1[4] j = θ, k = 3 Step 3 [1, 2, 3, 3, 5, 6] 3 >= 2, place 3 at nums1[3] Step 4 [1, 2, 2, 3, 5, 6] 2 >= 2, place 2 at nums1[2] j = -1, k = 1 Since $j < \theta$, we are done. Final merged array:

- Place the larger one at the last available position (nums1[k]).
- $\bullet \quad \text{Move pointers accordingly (decrease } \texttt{i} \text{ , } \texttt{j} \text{ , and } \texttt{k} \text{)}.$

```
python

Group & Edit

while j >= 0:
    nums1[k] = nums2[j]
    j -= 1
    k -= 1

If there are remaining elements in nums2, copy them to nums1.

This happens when nums1 already had the largest values in place, and nums2 still has elements left.
```