🎯 Question (LeetCode 88: Merge Sorted Array)

Given two sorted integer arrays `nums1` and `nums2`, merge `nums2` into `nums1` as one sorted array. The number of elements initialized in `nums1` and `nums2` are `m` and `n` respectively. `nums1` has enough space (size equal to `m + n`) to hold additional elements from `nums2`.

Example:

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

🏷️ Merge Sorted Arrays

📁 1. Definition and Purpose

- Merge two sorted arrays into one sorted array in-place.
- Commonly used in sorting algorithms, merging datasets, or maintaining ordered sequences.

📁 2. Syntax and Structure (Python)

```python
# nums1: first array with extra space at the end
# m: number of initialized elements in nums1
# nums2: second array
# n: number of elements in nums2
```

📁 3. Two Approaches

🧾 Approach 1: Brute Force

- Concatenate nums2 into nums1 and sort.

```python
def merge_bruteforce(nums1, m, nums2, n):
    nums1[m:] = nums2  # Step 1: Copy nums2 into nums1
    nums1.sort()       # Step 2: Sort the combined array
```

- **Time Complexity:** O((m+n) log(m+n))
- **Space Complexity:** O(1) (in-place)

🧾 Approach 2: Optimized (Two Pointers from End)

- Start from the end of both arrays to fill nums1 from back to front.
- Avoid overwriting elements in nums1.

📂 4. Optimized Pseudocode

```
i = m - 1         # pointer for last element in nums1 initialized portion
j = n - 1         # pointer for last element in nums2
k = m + n - 1     # pointer for last position in nums1 array
while j >= 0:
    if i >= 0 and nums1[i] > nums2[j]:
        nums1[k] = nums1[i]
        i -= 1
    else:
        nums1[k] = nums2[j]
        j -= 1
    k -= 1
```

📂 5. Python Implementation with Detailed Comments

```python
from typing import List

def merge(nums1: List[int], m: int, nums2: List[int], n: int) -> None:
    """
    Merge nums2 into nums1 in-place.
    """
    # Initialize pointers for nums1, nums2, and the last index
    i = m - 1       # Last initialized element in nums1
    j = n - 1       # Last element in nums2
    k = m + n - 1   # Last position in nums1 to fill

    # Traverse from the end to the beginning
    while j >= 0:
        if i >= 0 and nums1[i] > nums2[j]:
            nums1[k] = nums1[i]  # Place larger element at end
            i -= 1
        else:
            nums1[k] = nums2[j]  # Place element from nums2
            j -= 1
        k -= 1  # Move backwards in nums1

# Example Usage
nums1 = [1,2,3,0,0,0]
m = 3
nums2 = [2,5,6]
n = 3
merge(nums1, m, nums2, n)
print(nums1)  # Output: [1,2,2,3,5,6]
```

📂 6. Internal Working

- Filling from end ensures no overwriting.
- Two-pointer comparison chooses the largest element each iteration.
- Only modifies nums1 array in-place.

📂 7. Best Practices

- Always check bounds (i >= 0, j >= 0) to avoid index errors.
- Use the two-pointer optimized approach for large arrays.

📂 8. Related Concepts

- Two-pointer technique
- In-place array manipulation
- Merge step in Merge Sort

📂 9. Complexity Analysis

- **Optimized Approach:**
    - Time: O(m+n)
    - Space: O(1)
- **Brute Force:**
    - Time: O((m+n) log(m+n))
    - Space: O(1) in-place

📂 10. Practice and Application

- LeetCode: 88 Merge Sorted Array, 21 Merge Two Sorted Lists
- Used in merging sorted datasets in data pipelines.
- Efficient in-place operations in memory-sensitive applications.