🎯 Question (LeetCode 26: Remove Duplicates from Sorted Array)

Given a sorted integer array `nums`, remove the duplicates **in-place** such that each element appears only once. Return the new length of the array after duplicates have been removed. The relative order of the elements should be maintained.

Example:

- Input: nums = [1,1,2]
- Output: 2, nums = [1,2]
- Input: nums = [0,0,1,1,1,2,2,3,3,4]
- Output: 5, nums = [0,1,2,3,4]

🏷️ Remove Duplicates from Sorted Array

📂 1. Definition and Purpose

- Remove duplicate elements from a sorted array in-place.
- Maintains relative order of elements while reducing array size.

📂 2. Syntax and Structure (Python)

```
# nums: sorted list of integers
```

📂 3. Two Approaches

🧾 Approach 1: Brute Force

- Use a set to remove duplicates and convert back to sorted list.

```
def remove_duplicates_bruteforce(nums):
    nums[:] = sorted(set(nums))  # Remove duplicates and sort
    return len(nums)
```

- **Time Complexity:** O(n log n)
- **Space Complexity:** O(n)

🧾 Approach 2: Optimized (Two Pointers In-place)

- Use two pointers to overwrite duplicates in-place.
- Achieves O(1) extra space.

4. Optimized Pseudocode

```
i = 0  # slow pointer
for j in range(1, len(nums)):  # fast pointer
    if nums[j] != nums[i]:
        i += 1
        nums[i] = nums[j]
return i + 1  # new length
```

5. Python Implementation with Detailed Comments

```python
def remove_duplicates(nums: list[int]) -> int:
    """
    Remove duplicates in-place and return the new length.
    """
    if not nums:
        return 0  # Handle empty array

    i = 0  # slow pointer
    for j in range(1, len(nums)):  # fast pointer
        if nums[j] != nums[i]:  # Found a new unique element
            i += 1
            nums[i] = nums[j]  # Overwrite at slow pointer

    return i + 1  # New length including first element

# Example Usage
nums = [0,0,1,1,1,2,2,3,3,4]
new_length = remove_duplicates(nums)
print(new_length)  # Output: 5
print(nums[:new_length])  # Output: [0,1,2,3,4]
```

6. Internal Working

- Fast pointer scans the array.
- Slow pointer keeps track of position for unique elements.
- Overwrites duplicates in-place.

7. Best Practices

- Always check for empty array.
- Use two-pointer technique for memory efficiency.
- Maintain in-place to save space in large datasets.

8. Related Concepts

- Two-pointer technique
- In-place array manipulation

- Removing duplicates from sorted sequences

📂 9. Complexity Analysis

- **Optimized Approach:**
  - Time: O(n)
  - Space: O(1)
- **Brute Force Approach:**
  - Time: O(n log n)
  - Space: O(n)

📂 10. Practice and Application

- LeetCode: 26 Remove Duplicates from Sorted Array, 80 Remove Duplicates from Sorted Array II
- Used in preprocessing sorted datasets and cleaning data streams.