🎯 Question (LeetCode 80: Remove Duplicates from Sorted Array II)

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

Example:

- Input: nums = [1,1,1,2,2,3]
- Output: 5, nums = [1,1,2,2,3]
- Input: nums = [0,0,1,1,1,1,2,3,3]
- Output: 7, nums = [0,0,1,1,2,3,3]

🏷️ Remove Duplicates from Sorted Array II

📂 1. Definition and Purpose

- Remove duplicate elements from a sorted array in-place allowing at most two occurrences.
- Maintains relative order while controlling duplicate frequency.

📂 2. Syntax and Structure (Python)

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

📂 3. Two Approaches

🧾 Approach 1: Brute Force

- Count occurrences and create a new array with at most two copies.

```
def remove_duplicates_bruteforce(nums):
    from collections import Counter
    counts = Counter(nums)
    temp = []
    for num in nums:
        if counts[num] > 0:
            occur = min(counts[num], 2)
            temp.extend([num] * occur)
            counts[num] = 0
    nums[:len(temp)] = temp
    return len(temp)
```

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

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

- Use two pointers; allow duplicates only if previous element occurred less than twice.
- O(1) extra space.

📂 4. Optimized Pseudocode

```
i = 0  # pointer for position to insert
for num in nums:
    if i < 2 or num != nums[i-2]:
        nums[i] = num
        i += 1
return i  # new length
```

📂 5. Python Implementation with Detailed Comments

```python
def remove_duplicates(nums: list[int]) -> int:
    """
    Remove duplicates allowing at most two occurrences in-place.
    """
    i = 0  # slow pointer for insertion
    for num in nums:  # fast pointer iterating over all elements
        if i < 2 or num != nums[i-2]:  # Check if allowed (at most 2)
            nums[i] = num  # Place valid element
            i += 1  # Move slow pointer
    return i

# Example Usage
nums = [0,0,1,1,1,1,2,3,3]
new_length = remove_duplicates(nums)
print(new_length)  # Output: 7
print(nums[:new_length])  # Output: [0,0,1,1,2,3,3]
```

📂 6. Internal Working

- Slow pointer tracks where to insert next valid element.
- Compare with element 2 positions behind to ensure at most two occurrences.
- Overwrites array in-place without extra memory.

📂 7. Best Practices

- For sorted arrays, always compare with previous elements to limit duplicates.
- Use optimized in-place method to reduce space usage.
- Test edge cases like empty arrays or arrays with all duplicates.

📂 8. Related Concepts

- Two-pointer technique
- In-place array modification
- Duplicate frequency control

📂 9. Complexity Analysis

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

📂 10. Practice and Application

- LeetCode: 80 Remove Duplicates from Sorted Array II
- Useful for data streams, maintaining limited duplicates, or memory-sensitive preprocessing.