# Problem Understanding

You are given a **sorted array** nums. Your task is to **remove the duplicates in-place**, such that each element appears **only once**, and return the **new length** of the array.

* **You must not** use extra space.
* You must modify the input array **in-place** with O(1) extra memory.
* The first part of the array (up to the returned length) should contain the unique elements in the original relative order.

### Example:

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

Output: 5, nums = [0,1,2,3,4,\_,\_,\_,\_,\_]

## Optimized Java Solution (Two Pointers)

class Solution {

public int removeDuplicates(int[] nums) {

if (nums.length == 0) return 0;

int i = 0; // points to the last unique element

for (int j = 1; j < nums.length; j++) {

if (nums[j] != nums[i]) {

i++;

nums[i] = nums[j]; // place the next unique element

}

}

return i + 1; // new length = last unique index + 1

}

}

# Dry Run Using Table

### Input:

nums = [0,0,1,1,1,2,2,3,3,4]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Step | i (write ptr) | j (read ptr) | nums[j] | nums[i] | Action | Updated nums |
| 0 | 0 | 1 | 0 | 0 | Duplicate, skip | [0,0,...] |
| 1 | 0 | 2 | 1 | 0 | Unique → i++, nums[i] = 1 | [0,1,...] |
| 2 | 1 | 3 | 1 | 1 | Duplicate, skip | [0,1,...] |
| 3 | 1 | 4 | 1 | 1 | Duplicate, skip | [0,1,...] |
| 4 | 1 | 5 | 2 | 1 | Unique → i++, nums[i] = 2 | [0,1,2,...] |
| 5 | 2 | 6 | 2 | 2 | Duplicate, skip | [0,1,2,...] |
| 6 | 2 | 7 | 3 | 2 | Unique → i++, nums[i] = 3 | [0,1,2,3,...] |
| 7 | 3 | 8 | 3 | 3 | Duplicate, skip | [0,1,2,3,...] |
| 8 | 3 | 9 | 4 | 3 | Unique → i++, nums[i] = 4 | [0,1,2,3,4,...] |

✅ Final result: length = i + 1 = 5

Array becomes: [0, 1, 2, 3, 4, \_, \_, \_, \_, \_]

## Time & Space Complexity

| **Metric** | **Value** |
| --- | --- |
| Time | O(n) — one pass through the array |
| Space | O(1) — in-place, no extra memory |

# Alternate Approaches

## ****Using a Set (not allowed for this problem)****

* Store all values in a LinkedHashSet and rewrite array.
* **Not in-place**, uses extra space.
* Time: O(n), Space: O(n)
* ❌ Not acceptable for interview/code problem

## ****Two Pointer (Best Practice)****

* Read pointer (j): scans every element
* Write pointer (i): tracks where to place the next unique element
* ✅ Clean, fast, and in-place

## Quick Test Cases:

[] → 0

[1,1,1,1] → 1

[1,2,3,4] → 4

[0,0,1,1,2,2,3,3,4,4] → 5