LeetCode 75. Sort Colors

1. Problem Title & Link

- **75. Sort Colors**
- https://leetcode.com/problems/sort-colors/

2. Problem Statement (Short Summary)

Given an array nums with n objects colored red, white, or blue,
sort them **in-place** so that objects of the same color are adjacent, and colors are in the order:
0 → red, 1 → white, 2 → blue

3. Examples (Input → Output)

Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]

Input: nums = [2,0,1]
Output: [0,1,2]

4. Constraints

- n == nums.length
- 1 <= n <= 300
- nums[i] ∈ {0, 1, 2}

5. Thought Process (Step by Step)

This problem is not just sorting — it's about **in-place partitioning**.
You must reorder without using built-in sort and **O(1)** space.

Approach 1: Counting Sort (Simple but not elegant)

Count how many 0s, 1s, and 2s, then overwrite the array.

✅ Works but not *in-place* partition logic (less preferred for interviews).

✅ **Approach 2: Dutch National Flag Algorithm (Optimal & Beautiful 💡)**

We use **three pointers**:

- low → next position of 0
- mid → current index
- high → next position of 2

The array is divided conceptually into 4 regions:
[0..low-1] → all 0s
[low..mid-1] → all 1s
[mid..high] → unknowns (to process)
[high+1..n-1] → all 2s

Algorithm Steps:

1. Initialize low = 0, mid = 0, high = n-1
2. While mid <= high:
   - If nums[mid] == 0: swap nums[low] ↔ nums[mid], then low++, mid++
   - If nums[mid] == 1: just mid++
   - If nums[mid] == 2: swap nums[mid] ↔ nums[high], then high-- (don't increment mid yet)

This works because:

- 0s go to front (low)
- 2s go to end (high)
- 1s stay naturally in the middle

6. Pseudocode

```
low = 0
mid = 0
high = n - 1

while mid <= high:
  if nums[mid] == 0:
    swap(nums[low], nums[mid])
    low++, mid++
  else if nums[mid] == 1:
    mid++
  else:
    swap(nums[mid], nums[high])
    high--
```

7. Code Implementation

✅ **Python**

```
class Solution:
  def sortColors(self, nums: List[int]) -> None:
    low, mid, high = 0, 0, len(nums) - 1

    while mid <= high:
      if nums[mid] == 0:
        nums[low], nums[mid] = nums[mid], nums[low]
        low += 1
        mid += 1
      elif nums[mid] == 1:
        mid += 1
      else:
        nums[mid], nums[high] = nums[high], nums[mid]
        high -= 1
```

✅ **Java**

```
class Solution {
    public void sortColors(int[] nums) {
        int low = 0, mid = 0, high = nums.length - 1;

        while (mid <= high) {
            if (nums[mid] == 0) {
                int temp = nums[low];
                nums[low] = nums[mid];
                nums[mid] = temp;
                low++; mid++;
            } else if (nums[mid] == 1) {
                mid++;
            } else {
                int temp = nums[mid];
                nums[mid] = nums[high];
                nums[high] = temp;
                high--;
            }
        }
    }
}
```

8. Time & Space Complexity

- **Time:** O(n) — single traversal
- **Space:** O(1) — in-place

9. Dry Run (Step-by-Step Execution)

👉 Input: [2,0,2,1,1,0]

Step	low	mid	high	nums[mid]	Action	Array
1	0	0	5	2	swap mid↔high → high--	[0,0,2,1,1,2]
2	0	0	4	0	swap low↔mid → low++, mid++	[0,0,2,1,1,2]
3	1	1	4	0	swap low↔mid → low++, mid++	[0,0,2,1,1,2]
4	2	2	4	2	swap mid↔high → high--	[0,0,1,1,2,2]
5	2	2	3	1	mid++	[0,0,1,1,2,2]
6	2	3	3	1	mid++	[0,0,1,1,2,2]

✅ Output: [0,0,1,1,2,2]

10. Concept Insight Table

Core Concept	Common Use Cases	Common Traps	Builds / Next Steps
Dutch National Flag (3-way Partitioning) — partition array into 3 regions using constant space.	- 3-category problems (colors, negatives/zeros/positives) - QuickSort partition logic - In-place classification tasks	- Incrementing mid after swapping with high (wrong) - Mismanaging pointer order (low, mid, high) - Forgetting that array is modified in-place	🔷 Builds to **LeetCode 912 (Sort an Array)** (QuickSort partitioning) 🔷 Connects to **partition-based sorting** (like QuickSelect) 🔷 Reinforces **pointer invariants and state transitions**

11. Common Mistakes / Edge Cases

- Incrementing both pointers after swapping with high (causes missed elements).
- Forgetting in-place constraint (using extra arrays).
- Confusing with “count and overwrite” approach (works but less elegant).

12. Variations / Follow-Ups

- Sort array with **k colors** → use counting or generalized partitioning.
- Partition array by **negative / zero / positive** → same algorithm.
- Use it to understand **QuickSort partition step** deeply.