Problem Statement

You are given an array nums with n objects colored **red (0)**, **white (1)**, and **blue (2)**.

👉 Task: Sort them **in-place** so that all 0s come first, followed by 1s, then 2s.

- You must not use the library's sort.
- Expected time: O(n).
- Expected space: O(1).

Example

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

Approaches

Approach 1: Counting Sort (Simple but 2-pass)

1. Count how many 0, 1, 2.
2. Rewrite array.

- Time: O(n)
- Space: O(1)

Good for understanding but not optimal interview answer.

Approach 2: Dutch National Flag Algorithm (One-pass, Optimal)

We maintain **3 regions** using **3 pointers**:

- low → next position for 0
- mid → current element to check
- high → next position for 2

Rules:

Problem Statement

You are given an array nums with n objects colored **red (0)**, **white (1)**, and **blue (2)**.

👉 Task: Sort them **in-place** so that all 0s come first, followed by 1s, then 2s.

- You must not use the library's sort.
- Expected time: O(n).
- Expected space: O(1).

Example

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

Approaches

Approach 1: Counting Sort (Simple but 2-pass)

1. Count how many 0, 1, 2.
2. Rewrite array.

- Time: O(n)
- Space: O(1)

Good for understanding but not optimal interview answer.

Approach 2: Dutch National Flag Algorithm (One-pass, Optimal)

We maintain **3 regions** using **3 pointers**:

- low → next position for 0
- mid → current element to check
- high → next position for 2

Rules:

Code:

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

        while (mid <= high) {
            if (nums[mid] == 0) {
                swap(nums, low, mid);
                low++;
                mid++;
            } else if (nums[mid] == 1) {
                mid++;
            } else { // nums[mid] == 2
                swap(nums, mid, high);
                high--;
            }
        }
    }

    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
}
```

Dry Run Example

Input:

nums = [2, 0, 2, 1, 1, 0]
Initial pointers:

low = 0, mid = 0, high = 5

Iteration 1

- nums[mid] = 2
- Swap nums[mid] with nums[high] → swap(0,5)

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

low=0, mid=0, high=4

Iteration 2

- nums[mid] = 0
- Swap nums[mid] with nums[low] → swap(0,0) (no change)

nums = [0, 0, 2, 1, 1, 2]
low=1, mid=1, high=4

Iteration 3

- nums[mid] = 0
- Swap nums[mid] with nums[low] → swap(1,1) (no change)

nums = [0, 0, 2, 1, 1, 2]
low=2, mid=2, high=4

Iteration 4

- nums[mid] = 2
- Swap nums[mid] with nums[high] → swap(2,4)

nums = [0, 0, 1, 1, 2, 2]
low=2, mid=2, high=3

Iteration 5

- nums[mid] = 1
- Just move mid++

nums = [0, 0, 1, 1, 2, 2]
low=2, mid=3, high=3

Iteration 6

- nums[mid] = 1
- Just move mid++

nums = [0, 0, 1, 1, 2, 2]
low=2, mid=4, high=3

Now mid > high, loop ends ✅

Final sorted array:

[0, 0, 1, 1, 2, 2]

Teaching Points for Students

1. **3 zones**:
    - [0..low-1] → already 0s
    - [low..mid-1] → already 1s
    - [high+1..end] → already 2s
2. **Decisions** at each step:
    - If 0 → expand 0-zone (low++ + mid++)
    - If 1 → expand 1-zone (mid++)
    - If 2 → expand 2-zone (high--)
3. Runs in **O(n)**, in-place, no extra memory.