LeetCode 912: Sort an Array.

1. Problem Title & Link

- **912. Sort an Array**
- https://leetcode.com/problems/sort-an-array/

2. Problem Statement (Short Summary)

We are given an integer array nums. We need to return the array sorted in ascending order.

3. Examples (Input → Output)

Input: nums = [5,2,3,1]

Output: [1,2,3,5]

Input: nums = [5,1,1,2,0,0]

Output: [0,0,1,1,2,5]

4. Constraints

- 1 <= nums.length <= 5 * 10^4
- -5 * 10^4 <= nums[i] <= 5 * 10^4

5. Thought Process (Step by Step)

- **Brute Force**: Use built-in sort. Works, but in interviews they expect implementation.
- **Better**: Implement **Merge Sort** (O(n log n), stable, consistent).
- **Other Options**: QuickSort (avg O(n log n), worst $O(n^2)$), HeapSort (O(n log n), less stable).

6. Pseudocode (Language-Independent)

```
function mergeSort(nums):
    if length(nums) <= 1:
        return nums

    mid = len(nums) / 2
    left = mergeSort(nums[0:mid])
    right = mergeSort(nums[mid:])

    return merge(left, right)
```

```
function merge(left, right):
    result = []
    while both arrays have elements:
        pick smaller one → push to result
    append remaining elements
    return result
```

7. Code Implementation (Python)

```
class Solution:
    def sortArray(self, nums: List[int]) -> List[int]:
        def merge_sort(arr):
            if len(arr) <= 1:
                return arr
            mid = len(arr) // 2
            left = merge_sort(arr[:mid])
            right = merge_sort(arr[mid:])
            return merge(left, right)

        def merge(left, right):
            result = []
            i = j = 0
            while i < len(left) and j < len(right):
                if left[i] < right[j]:
                    result.append(left[i])
                    i += 1
                else:
                    result.append(right[j])
                    j += 1
            result.extend(left[i:])
            result.extend(right[j:])
            return result

        return merge_sort(nums)
```

8. Time & Space Complexity Analysis

- **Time:** O(n log n) (divide + merge)
- **Space:** O(n) (temporary arrays during merge)

9. Common Mistakes / Edge Cases

- Forgetting base case (len(arr) <= 1) → infinite recursion.
- Incorrect merge step → missing leftover elements.
- Confusing quicksort vs mergesort complexity.

10. Variations / Follow-Ups

- Implement **QuickSort** (partition based).
- Use **HeapSort** (priority queue).
- Sort **Linked List** instead of array.

11. Dry Run (Step by Step Execution)

👉 Input: [5, 2, 3, 1]

- Call merge_sort([5, 2, 3, 1])
  - Split → [5, 2] and [3, 1]

➡ First Half [5, 2]

- Call merge_sort([5, 2])
  - Split → [5] and [2]
  - Merge: compare 5 vs 2 → [2, 5]

➡ Second Half [3, 1]

- Call merge_sort([3, 1])
  - Split → [3] and [1]
  - Merge: compare 3 vs 1 → [1, 3]

➡ Final Merge

- Merge [2, 5] and [1, 3]
  - Compare 2 vs 1 → pick 1 → [1]
  - Compare 2 vs 3 → pick 2 → [1, 2]
  - Compare 5 vs 3 → pick 3 → [1, 2, 3]
  - Append leftover 5 → [1, 2, 3, 5]

✅ Final Sorted Output = [1, 2, 3, 5]