LeetCode 56. Merge Intervals

1. Problem Title & Link

- **56. Merge Intervals**
- https://leetcode.com/problems/merge-intervals/

2. Problem Statement (Short Summary)

We are given an array of intervals, where each interval = [start, end].

Merge **all overlapping intervals** and return the result as a list of non-overlapping intervals sorted by start time.

3. Examples (Input → Output)

Input: intervals = [[1,3],[2,6],[8,10],[15,18]]

Output: [[1,6],[8,10],[15,18]]

Input: intervals = [[1,4],[4,5]]

Output: [[1,5]]

4. Constraints

- 1 <= intervals.length <= 10^4
- intervals[i].length == 2
- 0 <= start_i <= end_i <= 10^4

5. Thought Process (Step by Step)

1. Sort intervals by **start time**.
2. Initialize result list with first interval.
3. Traverse remaining intervals:
    - If current interval overlaps with last added → merge.
    - Else → add as new interval.

6. Pseudocode (Language-Independent)

```
sort intervals by start
result = [intervals[0]]

for each interval in intervals[1:]:
    last = result[-1]
    if interval.start <= last.end:
        last.end = max(last.end, interval.end)    # merge
```

```
    else:
        result.append(interval)

return result
```

7. Code Implementation

✅ **Python**

```
class Solution:
    def merge(self, intervals: List[List[int]]) -> List[List[int]]:
        intervals.sort(key=lambda x: x[0])  # sort by start
        merged = [intervals[0]]

        for start, end in intervals[1:]:
            last_start, last_end = merged[-1]
            if start <= last_end:
                merged[-1][1] = max(last_end, end)
            else:
                merged.append([start, end])
        return merged
```

✅ **Java**

```
class Solution {
    public int[][] merge(int[][] intervals) {
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
        List<int[]> merged = new ArrayList<>();
        merged.add(intervals[0]);

        for (int i = 1; i < intervals.length; i++) {
            int[] last = merged.get(merged.size() - 1);
            int[] curr = intervals[i];
            if (curr[0] <= last[1]) {
                last[1] = Math.max(last[1], curr[1]);
            } else {
                merged.add(curr);
            }
        }
        return merged.toArray(new int[merged.size()][]);
    }
}
```

8. Time & Space Complexity Analysis

- Sorting: O(n log n)
- Merging: O(n)

- Total: **O(n log n)**
- Space: O(n) (for result list)

9. Common Mistakes / Edge Cases

- Forgetting to sort intervals first → incorrect merging.
- Not updating the merged interval end correctly.
- Single interval input → should return as is.

10. Variations / Follow-Ups

- Insert interval into sorted list (LeetCode 57).
- Count total merged intervals instead of returning them.
- Merge intervals in streaming data.

11. Dry Run (Step by Step Execution)

👉 Input:

intervals = [[1,3],[2,6],[8,10],[15,18]]

1. Sort by start → [[1,3],[2,6],[8,10],[15,18]] (already sorted).
2. Initialize result = [[1,3]].

- Compare [2,6] with [1,3]: overlap (2 ≤ 3).
  → merge = [1, max(3,6)] = [1,6].
  → result = [[1,6]].
- Compare [8,10] with [1,6]: no overlap (8 > 6).
  → add [8,10].
  → result = [[1,6],[8,10]].
- Compare [15,18] with [8,10]: no overlap (15 > 10).
  → add [15,18].
  → result = [[1,6],[8,10],[15,18]].

✅ Output:

[[1,6],[8,10],[15,18]]