LeetCode 435. Non-overlapping Intervals

1. Problem Title & Link

- **435. Non-overlapping Intervals**
- https://leetcode.com/problems/non-overlapping-intervals/

2. Problem Statement (Short Summary)

We are given an array of intervals.

We need to **remove the minimum number of intervals** so that the remaining intervals are **non-overlapping**.

Return the number of intervals we must remove.

3. Examples (Input → Output)

Input: intervals = [[1,2],[2,3],[3,4],[1,3]]

Output: 1

Explanation: Remove [1,3] → remaining are non-overlapping.

Input: intervals = [[1,2],[1,2],[1,2]]

Output: 2

Explanation: Remove two intervals → one left.

Input: intervals = [[1,2],[2,3]]

Output: 0

Explanation: Already non-overlapping.

4. Constraints

- 1 <= intervals.length <= 10^5
- intervals[i].length == 2
- -5 * 10^4 <= start_i < end_i <= 5 * 10^4

5. Thought Process (Step by Step)

This is a **greedy problem**.

- Sort intervals by **end time**.
- Always pick the interval with the **earliest end** (leaves room for others).
- If an interval overlaps with the chosen one → remove it.
- Count removals.

6. Pseudocode (Language-Independent)

```
sort intervals by end
count = 0
end = -∞

for each interval in intervals:
    if interval.start >= end:
        end = interval.end   # keep interval
    else:
        count += 1           # remove overlapping interval

return count
```

7. Code Implementation

✅ **Python**

```python
class Solution:
    def eraseOverlapIntervals(self, intervals: List[List[int]]) -> int:
        intervals.sort(key=lambda x: x[1])  # sort by end time
        count = 0
        end = float('-inf')

        for start, finish in intervals:
            if start >= end:
                end = finish  # keep it
            else:
                count += 1    # remove it
        return count
```

✅ **Java**

```java
class Solution {
    public int eraseOverlapIntervals(int[][] intervals) {
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[1], b[1]));
        int count = 0;
        int end = Integer.MIN_VALUE;

        for (int[] interval : intervals) {
            if (interval[0] >= end) {
                end = interval[1];  // keep interval
            } else {
                count++;  // remove overlapping
            }
        }
        return count;
    }
}
```

8. Time & Space Complexity Analysis

- Sorting: O(n log n)
- Scan: O(n)
- Total: **O(n log n)**
- Space: O(1)

9. Common Mistakes / Edge Cases

- Sorting by start time instead of end time → greedy fails.
- Forgetting to update end after keeping an interval.
- Handling negative start values incorrectly.

10. Variations / Follow-Ups

- Maximum number of non-overlapping intervals (just return kept count instead of removed).
- Interval scheduling problems in job scheduling.

11. Dry Run (Step by Step Execution)

👉 Input:

intervals = [[1,2],[2,3],[3,4],[1,3]]

1. Sort by end time → [[1,2],[2,3],[1,3],[3,4]]
2. Initialize: count = 0, end = -∞

- Interval [1,2]: start=1 ≥ -∞ → keep → end=2
- Interval [2,3]: start=2 ≥ 2 → keep → end=3
- Interval [1,3]: start=1 < 3 → overlap → remove → count=1
- Interval [3,4]: start=3 ≥ 3 → keep → end=4

✅ Final: count = 1

✅ Output:

1