LeetCode 452. Minimum Number of Arrows to Burst Balloons

1. Problem Title & Link

- **452. Minimum Number of Arrows to Burst Balloons**
- https://leetcode.com/problems/minimum-number-of-arrows-to-burst-balloons/

2. Problem Statement (Short Summary)

We are given points where each element is [start, end], representing the **horizontal diameter of a balloon**.

An arrow can be shot vertically at some x-coordinate.

- A balloon [start, end] bursts if start ≤ x ≤ end.
- We want to **burst all balloons with the minimum arrows**.

Return the minimum number of arrows needed.

3. Examples (Input → Output)

Input: points = [[10,16],[2,8],[1,6],[7,12]]
Output: 2
Explanation: Shoot at x = 6 (bursts [2,8],[1,6]) and x = 11 (bursts [10,16],[7,12]).

Input: points = [[1,2],[3,4],[5,6],[7,8]]
Output: 4
Explanation: Each balloon needs its own arrow.

Input: points = [[1,2],[2,3],[3,4],[4,5]]
Output: 2
Explanation: Shoot at x = 2 and x = 4.

4. Constraints

- 1 <= points.length <= 10^5
- points[i].length == 2
- -2^31 <= start_i < end_i <= 2^31 - 1

5. Thought Process (Step by Step)

This is similar to **interval scheduling** (like 435 Non-overlapping Intervals).

- We want to minimize arrows → maximize balloons per arrow.
- Sort balloons by **end coordinate**.
- Shoot an arrow at the end of the first balloon.
- For each balloon:
  - If it overlaps with the previous arrow position → no new arrow.
  - Else → need a new arrow.

6. Pseudocode (Language-Independent)

```
sort points by end
arrows = 1
end = points[0].end

for each balloon in points[1:]:
    if balloon.start > end:
        arrows += 1
        end = balloon.end

return arrows
```

7. Code Implementation

✅ **Python**

```
class Solution:
    def findMinArrowShots(self, points: List[List[int]]) -> int:
        points.sort(key=lambda x: x[1])  # sort by end
        arrows = 1
        end = points[0][1]

        for start, finish in points[1:]:
            if start > end:
                arrows += 1
                end = finish
        return arrows
```

✅ **Java**

```
class Solution {
    public int findMinArrowShots(int[][] points) {
            Arrays.sort(points, (a, b) -> Integer.compare(a[1],
b[1]));
        int arrows = 1;
        int end = points[0][1];

        for (int i = 1; i < points.length; i++) {
            if (points[i][0] > end) {
                arrows++;
                end = points[i][1];
            }
        }
        return arrows;
    }
}
```

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

- Forgetting to handle only **1 balloon** → should return 1.
- Using wrong condition (>= vs >).
  - If start == end, it can still be burst by the same arrow.
- Integer overflow when handling large coordinates (use long in Java if needed).

10. Variations / Follow-Ups

- Find the **x positions of arrows** (not just count).
- Extend problem to 2D (burst balloons in a plane).

11. Dry Run (Step by Step Execution)

👉 Input:

points = [[10,16],[2,8],[1,6],[7,12]]

1. Sort by end:
   → [[1,6],[2,8],[7,12],[10,16]]
2. Initialize: arrows = 1, end = 6

- Balloon [1,6]: covered → no new arrow.
- Balloon [2,8]: start=2 ≤ 6 → covered by same arrow.
- Balloon [7,12]: start=7 > 6 → need new arrow → arrows=2, end=12.
- Balloon [10,16]: start=10 ≤ 12 → covered.

✅ Final: arrows = 2