🏅 Relative Ranks – Full Documentation

Table of Contents

1. 📌 Problem Statement

You're given an array score of size n, where score[i] is the score of the i-th athlete in a competition. All scores are unique.

- Rank athletes based on their scores:
    - 1st: "Gold Medal"
    - 2nd: "Silver Medal"
    - 3rd: "Bronze Medal"
    - 4th to nth: rank as string of placement number (e.g., "4", "5", ...)

Return an array answer where answer[i] is the rank of the i-th athlete.

2. 💡 Intuition

We need to rank athletes based on performance but maintain their original order in the result. A common pattern for such problems is:

- Store original indices.
- Sort by value.
- Assign rankings and map them back to the original indices.

3. 🔍 Key Observations

- Sorting scores gives us the order of ranks.
- Mapping scores to their original index helps us retain the input order in the output.
- Only the top 3 ranks have custom labels; the rest are numeric.

4. ⚙ Approach

- Pair scores with indices using enumerate(score).
- Sort the list in descending order of scores.
- Initialize a result array with empty strings of the same size.
- Assign ranks:
  - Index 0: "Gold Medal"
  - Index 1: "Silver Medal"
  - Index 2: "Bronze Medal"
  - Others: string of (rank index + 1)
- Return the result list.

5. ⚠ Edge Cases

- Minimum input size: n = 1 → Should return ["Gold Medal"].
- Large scores or input size ($n = 10^4$) → Ensure efficient sorting.
- No duplicates → No need to handle tie-breaking.

6. □ Complexity Analysis

Time Complexity

- O(n log n) for sorting the scores.
- O(n) for assigning ranks.
- ✅ Overall: O(n log n)

Space Complexity

- O(n) for storing result and sorted score-index pairs.

7. 🔄 Alternative Approaches

- Using a max heap (priority queue):
  - Store negative of scores to simulate max-heap.
  - More complex and less efficient in Python than sorting.
- Counting sort: Not efficient here due to large score range (0 to 10^6) with only n elements.

8. □ Test Cases

✅ Test Case 1

Input: [5, 4, 3, 2, 1]
Output: ["Gold Medal", "Silver Medal", "Bronze Medal", "4", "5"]

✅ Test Case 2

Input: [10, 3, 8, 9, 4]
Output: ["Gold Medal", "5", "Bronze Medal", "Silver Medal", "4"]

✅ Test Case 3 – Minimum Size

Input: [99]
Output: ["Gold Medal"]

✅ Test Case 4 – Large Values

Input: [1000000, 999999, 888888]
Output: ["Gold Medal", "Silver Medal", "Bronze Medal"]

9. 🏁 Final Thoughts

- This problem is a classic sorting with index tracking task.
- Efficient and clean solution using enumerate() and sorting.
- Ideal for beginners to understand how to retain the original order while ranking or transforming elements.