506. Relative Ranks

Easy Array Sorting Heap (Priority Queue) Leetcode Link

Problem Description

You are given a list of scores for athletes in a competition. Each score in the `score` array is unique, and represents the score of an athlete, where `score[i]` is the score for the `i`th athlete. The goal is to rank the athletes based on their scores in descending order — the highest score gets the first rank, the second-highest gets the second rank, and so forth.

For the top three ranks, specific strings are used instead of numbers:

- The `1st` place is given the rank "Gold Medal".
- The `2nd` place is given the rank "Silver Medal".
- The `3rd` place is given the rank "Bronze Medal".

For all other positions starting from the `4th` to the `n`th place, the rank is the numeric place of the athlete in the sorted list. The task is to return an array where each element represents the rank of the corresponding athlete in the original input array.

Intuition

The problem can be simplified to sorting the athletes based on their scores and then assigning ranks according to their sorted positions. Since we are interested in the order of scores and need to remember the original indices, we can sort the indexes of the athletes based on the scores they map to. We perform the sort in descending order to ensure that the athlete with the highest score gets the first index.

Once we have the indices sorted by the athletes' scores, we can map these to their respective ranks. For the first three athletes (indices `0`, `1`, and `2`) after sorting, we assign the special ranks ("Gold Medal", "Silver Medal", "Bronze Medal"). For all other athletes, we convert their index position (plus one, since we rank starting from `1` not `0`) to a string to represent their numeric rank. The reason we add `1` is that ranks are typically 1-indexed, not 0-indexed. Finally, we place these rank strings in a new array `ans` with the same size as the input array, ensuring that each athlete's rank is placed in their original index in the input array.

Solution Approach

The provided Python code implements a solution that carefully maps the athletes' scores to their respective ranks. Here's how the code works in detail, breaking it down into key steps:

1. First, we calculate the length of the input `score` array, which represents the number of athletes. This is stored in the variable `n`.
2. We then create a list called `idx`, which is a list of indices from `0` to `n-1`. These indexes correspond to the positions of athletes in the `score` array.
3. The `idx` list is sorted using a custom `key` function which sorts the indexes based on the scores at those indexes in descending order. The lambda function `lambda x: -score[x]` returns the negative score value for each index, thus ensuring a descending sort.
4. A `top3` list is defined with the strings "Gold Medal", "Silver Medal", and "Bronze Medal" which are the ranks for the top three athletes.
5. We create an array `ans` initialized with `None` values, with the same length as the number of athletes. This will be used to store the final rankings.
6. A loop runs from `0` to `n` to assign ranks to all athletes. The current index in the sorted `idx` list determines the athlete's placement in the sorted order.
7. Inside the loop:
   - If the index is less than `3` (meaning the athlete is in the top three), we use the `top3` list to assign "Gold Medal", "Silver Medal", or "Bronze Medal" accordingly.
   - For all other indices (4th place and beyond), we convert the index to a string, while also adding `1` to it to account for the fact that rankings start from `1` instead of `0`. This string is the athlete's rank.
8. We assign these ranks to the corresponding positions in the `ans` array based on the sorted indices. This ensures each athlete's rank is placed at the same index as they originally appeared in the `score` array.
9. Finally, the `ans` array, now filled with the rankings, is returned.

This approach ensures that the athletes are ranked according to their scores, with the original array structure preserved, and special rankings are given to the top three athletes.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach described in the content above. Suppose we have the following list of scores for a competition:

```
scores = [50, 30, 80, 70, 10]
```

We want to determine the ranks of these athletes, with the top three receiving "Gold Medal", "Silver Medal", and "Bronze Medal", respectively.

The detailed steps would be:

1. Calculate the length of the scores array, `n = 5`.
2. Create a list of indices `idx = [0, 1, 2, 3, 4]`.
3. Sort the `idx` list by the athletes' scores in descending order:

   You will end up with indices sorted as `[2, 3, 0, 1, 4]` because when mapping these indices back to the `scores`, you get `[80, 70, 50, 30, 10]` which is in descending order.

4. Define a list `top3 = ["Gold Medal", "Silver Medal", "Bronze Medal"]`.
5. Initialize an array `ans = [None, None, None, None, None]` to store the rankings.
6. Loop through indices in `idx` and assign ranks as follows:
   - At `idx[0]` (which is 2), the score is 80. Since it is the highest, assign "Gold Medal" to `ans[2]`.
   - At `idx[1]` (which is 3), the score is 70. Assign "Silver Medal" to `ans[3]`.
   - At `idx[2]` (which is 0), the score is 50. Assign "Bronze Medal" to `ans[0]`.
   - At `idx[3]` (which is 1), the score is 30. This is the 4th highest score, so assign the string '4' to `ans[1]`.
   - At `idx[4]` (which is 4), the score is 10. This is the 5th highest score, so assign the string '5' to `ans[4]`.

   The iteration assigns ranks based on the sorted indices and maps them back to the original `ans` array at their respective positions.

7. Final `ans` array after the loop will be:

```
ans = ["Bronze Medal", "4", "Gold Medal", "Silver Medal", "5"]
```

As shown in the `ans` array, we have assigned the correct rank to each athlete based on their original position in the `scores` array. Athlete with score 80 (3rd in the original array) gets "Gold Medal", athlete with score 70 (4th in the original array) gets "Silver Medal", and so on.

The final output correctly represents the ranks of the athletes corresponding to their scores in descending order, fulfilling the problem description.

Python Solution

```python
class Solution:
    def find_relative_ranks(self, scores: List[int]) -> List[str]:
        # Find the number of scores
        num_scores = len(scores)

        # Create a list of indices from the scores list
        indices = list(range(num_scores))

        # Sort indices based on the scores in descending order
        indices.sort(key=lambda x: -scores[x])

        # Define strings for the top three positions
        top_three_medals = ['Gold Medal', 'Silver Medal', 'Bronze Medal']

        # Initialize the answer list with placeholders
        answer = [None] * num_scores

        # Fill the answer list with the appropriate rank strings
        for rank, index in enumerate(indices):
            # If the rank is less than 3, assign a medal string
            if rank < 3:
                answer[index] = top_three_medals[rank]
            else:
                # Otherwise, assign the numeric rank as a string
                answer[index] = str(rank + 1)

        # Return the completed answer list with ranks
        return answer
```

Java Solution

```java
class Solution {

    // Method to find the relative ranks for athletes based on their scores.
    public String[] findRelativeRanks(int[] scores) {
        // Get the number of athletes based on the length of scores array.
        int numAthletes = scores.length;

        // Create a wrapper array to hold the indices of the scores.
        Integer[] indices = new Integer[numAthletes];

        // Initialize the indices array with values from 0 to numAthletes-1.
        for (int i = 0; i < numAthletes; ++i) {
            indices[i] = i;
        }

        // Sort the indices array based on the scores in descending order.
        // The comparison function uses the scores at the indices to sort the indices.
        Arrays.sort(indices, (a, b) -> Integer.compare(scores[b], scores[a]));

        // Create an array to hold the answers (relative ranks as strings).
        String[] ranks = new String[numAthletes];

        // Create an array to hold the medals for the top 3 athletes.
        String[] medals = new String[] {"Gold Medal", "Silver Medal", "Bronze Medal"};

        // Assign the appropriate rank to each athlete in the ranks array.
        for (int i = 0; i < numAthletes; ++i) {
            // If the rank is within the top 3, assign the corresponding medal.
            if (i < 3) {
                ranks[indices[i]] = medals[i];
            } else {
                // Otherwise, assign the rank number as a string (1-indexed).
                ranks[indices[i]] = String.valueOf(i + 1);
            }
        }

        // Return the array of ranks.
        return ranks;
    }
}
```

C++ Solution

```cpp
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

class Solution {
public:
    // This method finds and returns the relative ranks of athletes based on their scores.
    vector<string> findRelativeRanks(vector<int>& scores) {
        int numOfScores = scores.size();

        // Use a vector of pairs to keep track of the scores and their original indices.
        vector<pair<int, int>> indexedScores;
        for (int i = 0; i < numOfScores; ++i) {
            indexedScores.push_back(make_pair(scores[i], i));
        }

        // Sort the indexed scores in descending order using a custom comparator lambda function.
        sort(indexedScores.begin(), indexedScores.end(),
             [](const pair<int, int>& a, const pair<int, int>& b) { return a.first > b.first; });

        // Initialize the answer vector with the same size as the number of scores.
        vector<string> ranks(numOfScores);

        // Define medals for the top 3 scores.
        vector<string> medals = {"Gold Medal", "Silver Medal", "Bronze Medal"};

        // Assign ranks to the athletes based on sorted order.
        for (int i = 0; i < numOfScores; ++i) {
            if (i < 3) {
                // Assign medals to the top 3 scores.
                ranks[indexedScores[i].second] = medals[i];
            } else {
                // For others, assign the rank as a string (1-indexed).
                ranks[indexedScores[i].second] = to_string(i + 1);
            }
        }

        return ranks;
    }
};
```

Typescript Solution

```typescript
// Import the necessary functionalities from corresponding modules
import { sort } from 'algorithm';
import { vector, pair, make_pair } from 'vector';
import { string } from 'string';

// Define a function to find and return the relative ranks of athletes based on their scores.
function findRelativeRanks(scores: number[]): string[] {
    const numOfScores: number = scores.length;

    // Create an array to keep track of the scores along with their original indices.
    const indexedScores: Array<pair<number, number>> = [];
    for (let i = 0; i < numOfScores; ++i) {
        indexedScores.push(make_pair(scores[i], i));
    }

    // Use a custom comparator function for sorting the indexed scores in descending order.
    indexedScores.sort((a: pair<number, number>, b: pair<number, number>): number => b.first - a.first);

    // Prepare an array to hold the rank strings, initially empty.
    const ranks: string[] = new Array<string>(numOfScores);

    // Define medal strings for the top 3 ranks.
    const medals: string[] = ["Gold Medal", "Silver Medal", "Bronze Medal"];

    // Loop through the sorted scores to assign ranks or medals.
    for (let i = 0; i < numOfScores; ++i) {
        if (i < 3) {
            // Assign medals to the top 3 athletes.
            ranks[indexedScores[i].second] = medals[i];
        } else {
            // For the other athletes, assign their rank as a string (1-indexed).
            ranks[indexedScores[i].second] = (i + 1).toString();
        }
    }

    return ranks;
}

// The TypeScript syntax requires type annotations for parameters and variables.
// The standard JavaScript Array and sort functions are used in place of std::vector and std::sort.
// Note: TypeScript does not have an equivalent of std::pair, but you can use tuples or define an interface.

// Note: The above code block is written with the assumption that the TypeScript environment would mimic C++ functionality and modu
// In a standard TypeScript environment, the `algorithm` or `vector` modules do not exist, and JavaScript's array methods would be
```

Time and Space Complexity

The time complexity of the provided `findRelativeRanks` function is `O(n * log n)` where `n` is the number of athletes (the length of the input `score` list). This is because the function is performing a sort operation on a list of `n` indices, which is the most time-consuming part of the algorithm. The sorting function itself has a time complexity of `O(n log n)`.

After sorting, the function traverses the sorted indices once, creating a result list of ranks. This traversal is linear with respect to the number of athletes, giving it a time complexity of `O(n)`. However, this linear traversal does not dominate the sorting complexity, so the overall time complexity remains `O(n log n)`.

The space complexity of the algorithm is `O(n)`. We have `idx` and `ans` lists, each of size `n`. The `top3` list is of constant size and does not scale with `n`, so it does not affect the asymptotic space complexity.