1619. Mean of Array After Removing Some Elements

Easy Array Sorting Leetcode Link

Problem Description

The problem requires writing a function that calculates the trimmed mean of an array of integers. To clarify, the trimmed mean is the average obtained after removing the smallest and the largest 5% of the elements from the list. By trimming these outliers, the mean becomes a more representative measure of the central tendency for the data set, particularly when extreme values might skew the average.

For the mean to be accepted, it should be close to the actual mean within a margin of `10^-5`. This means the solution should be accurate enough up to the fifth decimal place.

Intuition

The process to achieve the trimmed mean can be broken down into the following steps:

1. **Sorting**: Sorting the array is a crucial first step. It's necessary because to easily identify and remove the smallest and largest 5% of elements, these elements need to be at the start and end of the list, respectively.
2. **Calculating Indices**: Next, we calculate the indices that represent the 5% mark from both the start and the end of the sorted array. This enables us to pinpoint where to slice the array to remove the unwanted elements. These indices are derived by multiplying the total length by 0.05 and 0.95, then converting the result to an integer. In Python, this truncates the decimal, effectively implementing a floor function and ensuring we get the correct slice for the desired percentages.
3. **Slicing**: After sorting the array and calculating the correct indices, the next step is to slice the array to remove the smallest and largest 5% of its elements. The sliced part is the range of elements we are interested in for calculating the trimmed mean.
4. **Mean Calculation and Rounding**: The final step involves calculating the mean of the remaining elements and rounding it to five decimal places as to conform with the problem's requirement for precision.

Note that the solution's accuracy hinges on proper rounding, and the `round()` function in Python serves to round the final result to the required number of decimal places.

This approach is intuitive and efficient, taking advantage of Python's built-in functions for sorting and arithmetic operations.

Solution Approach

The implementation of the solution uses a straightforward approach leveraging Python's list and built-in functions. Here's a breakdown of the key components of the solution:

1. **Sorting (`arr.sort()`)**: The first major step is to sort the array using the list's `.sort()` method. Sorting reorders the elements from the lowest to the highest values, which is essential for trimming the smallest and largest elements efficiently.
2. **Calculating Indices (`int(n * 0.05)`, `int(n * 0.95)`)**: The `n * 0.05` and `n * 0.95` calculations determine the indices for slicing the array. Given `n` is the total number of elements in the array, multiplying it by `0.05` gives the number of elements that represents the bottom 5%, and multiplying by `0.95` gives us the index just past the top 95% (effectively the top 5% mark). Casting the result to an `int` ensures we're working with whole index numbers.
3. **Slicing (`arr[start:end]`)**: With the start and end indices now established, we can slice the sorted array. The slicing operation `arr[start:end]` removes the first 5% and the last 5% of the array, which are the smallest and largest values, respectively.
4. **Mean Calculation (`sum(t) / len(t)`) and Rounding (`round(…, 5)`)**: The trimmed array `t` is then passed to the `sum()` function, which adds up all the remaining values. This sum is then divided by `len(t)`, the count of elements in the trimmed array, to get the mean. Finally, the mean is rounded to five decimal places using the `round()` function to satisfy the precision requirement of the problem.

The choice to use Python's list sorting and slicing is based on their efficiency and ease of use. Sorting takes `O(n log n)` time complexity and slicing has `O(k)` complexity, where `k` is the number of elements to be copied (the size of the trimmed array in this case). The remaining operations (calculating the sum, finding the length of the list, and dividing) have linear time complexities, resulting in an overall efficient solution.

This solution uses no additional data structures, making it space-efficient as well, with the primary space usage being the input array and the trimmed slice.

Example Walkthrough

Let's use a small example to illustrate the solution approach. Suppose we are given the following array of integers:

```python
arr = [4, 8, 6, 5, 3, 2, 7, 9, 1, 0]
```

Our goal is to calculate the trimmed mean after removing the smallest and largest 5% of the elements. Since our array has 10 elements, 5% of this array corresponds to a single element at each end (since 10 * 0.05 = 0.5, which we floor to 0 elements, and we can't remove less than one whole element).

Following the steps as per the solution approach:

1. **Sorting (`arr.sort()`)**: First, we sort the array to get:

   ```python
   arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
   ```

   This puts the elements in ascending order.

2. **Calculating Indices (`int(n * 0.05)`, `int(n * 0.95)`)**: For our 10-element array, `n * 0.05` is `0.5`. As an integer, it's rounded down to `0`, representing the first element to remove. The second index is `n * 0.95`, which is `9.5`, and as an integer, it gets rounded down to `9`.

3. **Slicing (`arr[start:end]`)**: According to the calculated indices, we slice the sorted array from index `1` to index `9`:

   ```python
   t = arr[1:9]  # Sliced array is [1, 2, 3, 4, 5, 6, 7, 8]
   ```

   This removes the smallest and largest elements, `0` and `9`.

4. **Mean Calculation (`sum(t) / len(t)`) and Rounding (`round(…, 5)`)**: We then calculate the sum of the trimmed array `t` and divide by its length to get the mean:

   ```python
   mean = sum([1, 2, 3, 4, 5, 6, 7, 8]) / len([1, 2, 3, 4, 5, 6, 7, 8])  # mean is 4.5
   ```

   Then we round the mean to five decimal places:

   ```python
   trimmed_mean = round(mean, 5)  # trimmed_mean is 4.5, since there are no additional decimal places
   ```

Thus, for our example array, the trimmed mean after removing the smallest and largest 5% of the elements is 4.5. Please note that in cases where there are additional decimal places, the `round()` function will adjust the trimmed mean to five decimal places to meet the accuracy requirements.

Python Solution

```python
from typing import List

class Solution:
    def trimMean(self, arr: List[int]) -> float:
        # Calculate the number of elements in the array
        num_elements = len(arr)

        # Determine the indices to trim 5% of elements from both ends
        start_index = int(num_elements * 0.05)
        end_index = int(num_elements * 0.95)

        # Sort the array in non-decreasing order
        arr.sort()

        # Trim 5% of elements from each end of the sorted array
        trimmed_arr = arr[start_index:end_index]

        # Calculate the mean of the trimmed array
        trimmed_mean = sum(trimmed_arr) / len(trimmed_arr)

        # Round the mean to five decimal places and return it
        return round(trimmed_mean, 5)

```

Java Solution

```java
import java.util.Arrays; // Import Arrays class from the java.util package for sorting

class Solution {

    // Function to calculate the trimmed mean of an array after removing the smallest
    // and largest 5% of elements
    public double trimMean(int[] arr) {
        // Sort the input array
        Arrays.sort(arr);

        // Calculate the total number of elements in the array
        int totalElements = arr.length;

        // Calculate the number of elements to trim from each end (5% from both ends)
        int elementsToTrim = (int) (totalElements * 0.05);

        // Initialize the sum of the remaining elements
        double sum = 0;

        // Loop through the array from the first element after trimming
        // to the last element before trimming
        for (int index = elementsToTrim; index < totalElements - elementsToTrim; ++index) {
            // Add the current element to the sum
            sum += arr[index];
        }

        // Calculate the trimmed mean by dividing the sum of the remaining elements
        // by the number of elements after trimming (which is 90% of the total)
        return sum / (totalElements * 0.9);
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <algorithm> // Include algorithm header for sorting

class Solution {
public:
    // Function to calculate the trimmed mean of an array
    double trimMean(vector<int>& arr) {
        // Sort the array in non-decreasing order
        sort(arr.begin(), arr.end());

        int numElements = arr.size(); // Total number of elements in the array
        double sum = 0; // Initialize sum to store the sum of the elements

        // Calculate the starting index after trimming 5% from the front
        int startIndex = static_cast<int>(numElements * 0.05);
        // Calculate the ending index before trimming 5% from the back
        int endIndex = numElements - startIndex;

        // Loop through the array excluding the trimmed 5% from both ends
        for (int i = startIndex; i < endIndex; ++i) {
            sum += arr[i]; // Add the current element to the sum
        }

        // Calculate the trimmed mean by dividing sum by the number of elements after trimming
        double trimmedMean = sum / (numElements * 0.9);

        return trimmedMean; // Return the calculated trimmed mean
    }
};

```

Typescript Solution

```typescript
/**
 * Calculates the trimmed mean of an array after removing the smallest and
 * largest 5% of elements.
 *
 * @param {number[]} arr - The array of numbers from which to calculate the mean.
 * @return {number} - The trimmed mean of the array.
 */
function trimMean(arr: number[]): number {
    // Sort the array in ascending order.
    arr.sort((a, b) => a - b);

    // Calculate the number of elements to remove from each end of the array.
    let length = arr.length;
    let removeLength = Math.floor(length * 0.05);

    // Sum the array elements while excluding the top and bottom 5% of elements.
    let sum = 0;
    for (let i = removeLength; i < length - removeLength; i++) {
        sum += arr[i];
    }

    // Calculate the trimmed mean by dividing the sum by the number of elements included in the calculation.
    return sum / (length - 2 * removeLength);
}

```

Time and Space Complexity

Time Complexity

The time complexity of the given code is mainly determined by the sorting function. The `sort()` function in Python uses the Timsort algorithm, which has a time complexity of `O(n log n)` for sorting an array. Here's the breakdown:

- Sorting the array: `O(n log n)`
- Slicing the sorted array to remove the 5% of the elements from both ends: `O(n)`.

Therefore, the overall time complexity is `O(n log n + n)`. Since `O(n log n)` is the dominating term, the time complexity simplifies to `O(n log n)`.

Space Complexity

As for the space complexity:

- The array `t` that stores the sliced part of the original array introduces additional space. The space taken by `t` is proportional to the length of the slice, which is 90% of the original array, so `O(n)`.
- The `sort()` function may require `O(n)` space to perform the sorting.

The overall space complexity is `O(n)` since both the additional array and the sorting space complexity are linear with the input size.