Sorting before Two Pointer

The **total time complexity** is derived by analyzing each step of the algorithm and summing their individual complexities. Here's a detailed breakdown:

Steps of the Algorithm

1. **Sorting the Array (`arr.sort()`):**
   - The `sort()` function sorts the array in ascending order.
   - Sorting an array of size `n` takes **O(n log n)** using efficient sorting algorithms like Timsort (used in Python).
2. **Two-Pointer Traversal (the `while` loop):**
   - After sorting, the two-pointer technique is applied.
   - The two pointers (`left` and `right`) traverse the array at most once. In each iteration:
     - Either `left` is incremented, or `right` is decremented.
     - This ensures that the loop runs for at most **O(n)** iterations.

Combining the Steps

- Sorting takes **O(n log n)**.
- The two-pointer traversal takes **O(n)**.
- Since these steps are performed sequentially (not nested), their complexities are **added**:
  Total Time Complexity=O(nlogn)+O(n)\text{Total Time Complexity} = O(n \log n) + O(n)
  Total Time Complexity=O(nlogn)+O(n)

Simplifying the Complexity

- In Big O notation, only the **dominant term** matters as `n` grows large.
  - **O(n log n)** dominates **O(n)** because logarithmic growth adds a significant factor to linear growth.
- Therefore, the total complexity simplifies to: O(nlogn)O(n \log n)O(nlogn)

Why Approximation?

The symbol `≈` in **O(n log n) + O(n) ≈ O(n log n)** indicates that:

- The **O(n)** term is negligible compared to **O(n log n)** for large `n`.
- So the effective time complexity is considered **O(n log n)**.

Practical Example

Let’s assume `n = 1,000,000`:

- **Sorting:** $n \log n = 1,000,000 \cdot \log_2 1,000,000 \approx 20,000,000$ operations.
- **Traversal:** $O(n) = 1,000,000$ operations.
- Clearly, the sorting step dominates.

Revision #1
Created 28 December 2024 00:32:20 by victor
Updated 28 December 2024 00:32:39 by victor