Task 4: Analysis of Anagram Detection Algorithm

1. Problem Definition

The objective is to determine if two given stringsare anagrams. Anagrams are strings that are composed of the exact same characters with the exact same frequencies, but potentially in a different order (e.g., "listen" and "silent").

2. Chosen Algorithm: Frequency Array

The chosen solution is a character counting (or frequency array) method. This algorithm achieves linear time complexity.

The core principle is to map each character to an index in an array. By counting the character frequencies in both strings, a direct comparison can be made in linear time.

3. Implementation Details

The algorithm is implemented as follows:

1. Array Initialization: An integer array, check, of size 128 is initialized to all zeros. This size is chosen to map all 128 standard ASCII characters (from 0 to 127), using the character's ASCII value directly as the index.
2. First Pass (s1): The algorithm iterates through the first string. For each character its corresponding position in the check array is incremented by one.
3. Second Pass (s2): The algorithm iterates through the second string. For each character, s2[i], its corresponding position in the check array is decremented by one.
4. Verification Pass: A final loop iterates through the entire check array (from index 0 to 127).
   - If a non-zero value is found, it signifies a mismatch in character frequencies between both strings. The function immediately returns false.
   - If the loop completes and all values are zero, it confirms that both strings have identical character frequencies, and the function returns true.

4. Performance and Complexity Analysis

This implementation includes key optimizations and exhibits high performance.

Optimizations:

- Length Check: The function first performs an essential pre-emptive check. This operation immediately rules out non-anagrams of different lengths, preventing unnecessary computation.
- Early Exit: During the final verification pass, the loop terminates and returns false as soon as the first non-zero count is detected, avoiding redundant checks.

Complexity Analysis:

- The algorithm achives a total compelxity of O(N).
- Space Complexity: The algorithm requires an auxiliary array of size k. As k is a constant (128), the space required does not grow with the input string size N. Therefore, the space complexity is O(1) (constant space).