LeetCode 242. Valid Anagram

1. Problem Title & Link

- **242. Valid Anagram**
- 🔗 https://leetcode.com/problems/valid-anagram/

2. Problem Statement (Short Summary)

Given two strings s and t, return **true** if t is an **anagram** of s, and **false** otherwise.

An *anagram* means both strings contain the same characters with the same frequency, but possibly in a different order.

3. Examples (Input → Output)

Input: s = "anagram", t = "nagaram"

Output: true

Input: s = "rat", t = "car"

Output: false

4. Constraints

- 1 <= s.length, t.length <= $5 * 10^4$
- s and t consist of lowercase English letters.

5. Thought Process (Step by Step)

There are **two intuitive approaches**, both great for students 💡

🧩 Approach 1: Sorting Based (Simple & Clear)

- Sort both strings.
- Compare if they're identical.

✅ Simple logic, but sorting takes O(n log n).

🧩 Approach 2: Frequency Count (Optimized)

- Count frequency of each character in both strings using a hash map (or fixed 26-size array for lowercase English).
- Compare frequencies.

✅ O(n) time, O(1) extra space (constant 26 characters).

✅ Perfect for interviews and teaching hashmap logic.

6. Pseudocode

```
if len(s) != len(t):
    return False

count = [0] * 26

for i in range(len(s)):
    count[ord(s[i]) - ord('a')] += 1
    count[ord(t[i]) - ord('a')] -= 1

for c in count:
    if c != 0:
        return False
return True
```

7. Code Implementation

✅ **Python**

```
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        if len(s) != len(t):
            return False

        count = [0] * 26
        for i in range(len(s)):
            count[ord(s[i]) - ord('a')] += 1
            count[ord(t[i]) - ord('a')] -= 1

        for val in count:
            if val != 0:
                return False
        return True
```

✅ **Java**

```
class Solution {
    public boolean isAnagram(String s, String t) {
        if (s.length() != t.length()) return false;

        int[] count = new int[26];
        for (int i = 0; i < s.length(); i++) {
            count[s.charAt(i) - 'a']++;
            count[t.charAt(i) - 'a']--;
        }
```

```
        for (int c : count)
            if (c != 0) return false;
        return true;
    }
}
```

8. Time & Space Complexity

Approach	Time	Space	Note
Sorting	O(n log n)	O(1) or O(n)	Simple but slower
Counting (Optimized)	O(n)	O(1)	Fastest, constant space for 26 letters

9. Dry Run (Step-by-Step Execution)

👉 Input:
s = "anagram"
t = "nagaram"

Step	s[i]	t[i]	count changes (net effect)
1	a	n	a:+1, n:-1
2	n	a	n:0, a:0
3	a	g	g:-1
4	g	a	g:0
5	r	r	r:0
6	a	a	a:0
7	m	m	m:0

✅ All counts = 0 → return True

10. Concept Insight Table

Core Concept	Common Use Cases	Common Traps	Builds / Next Steps
Character Frequency Counting — check if two sequences have identical frequency distribution.	- String anagram validation - Permutation checks - Frequency histogram comparison	- Forgetting to check string length first - Not resetting count array correctly - Using hash map unnecessarily for small char sets	🔷 Builds to **LC 49 (Group Anagrams)** 🔷 Connects to **LC 383 (Ransom Note)** 🔷 Forms the base of **count-array patterns**

11. Common Mistakes / Edge Cases

- Not checking equal length → immediate false.
- Using `Counter` from Python directly (works, but hides logic).
- Case sensitivity — only lowercase letters assumed here.

12. Variations / Follow-Ups

- **LC 49:** Group all anagrams together.
- **LC 383:** Can one string be constructed from another?
- **LC 567:** Check if one string's permutation exists in another (sliding window + frequency array).