Valid Anagram

# Attempts	3
Date Solved	@October 21, 2025
Difficulty	Easy
Next Review	@October 28, 2025
Status	Solved
Topic/Pattern	Array and String

Link → https://neetcode.io/problems/is-anagram

Problem

- Given two strings `s` and `t`, return **True** if `t` is an anagram of `s`, otherwise **False**.
- An anagram means both strings contain the same characters with the same frequency, just in different order.

Examples

Input	Output	Reason
`s = "anagram"`, `t = "nagaram"`	True	Both have the same letters
`s = "rat"`, `t = "car"`	False	Different letters

Approach 1 — Sorting (Simple & Clean)

- **Idea:** If two strings are anagrams, their sorted versions will be identical.

```
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
```

```
        return sorted(s) == sorted(t)
```

- **Time Complexity:** O(n log n) — sorting dominates
- **Space Complexity:** O(1) or O(n) depending on sorting implementation
- Very simple, but slightly slower due to sorting.

Approach 2 — Hash Map / Dictionary (Optimal)

- **Idea:** Count frequency of each character in both strings and compare.

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

        countS, countT = {}, {}

        for ch in s:
            countS[ch] = countS.get(ch, 0) + 1

        for ch in t:
            countT[ch] = countT.get(ch, 0) + 1

        return countS == countT

#.get(key, 0) ka matlab hota hai:
# If key present in the dict then return the value it not return 0
# .get(key, 0) + 1 → adds the char in dictionary and increments the value by 1
```

- **Time Complexity:** O(n)
- **Space Complexity:** O(1) (since only 26 lowercase letters) or O(n) for general case
- Fastest and most scalable. Great for explaining hashing and frequency counting in interviews.

Edge Cases

- `s = ""`, `t = ""` → True (both empty)
- `s = "a"`, `t = "b"` → False
- Case sensitivity: `"a"` vs `"A"` → False unless handled
- Unicode characters → still works using Python's dict

Summary

Approach	Time	Space	Notes
Sorting	O(n log n)	O(1)	Easy, short solution
Hash Map	O(n)	O(1)/O(n)	Best for large input