Valid Anagram - Revision Notes

Problem Statement

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

Key Observations

- An anagram means both strings must have the same characters with the same frequency.
- If `s.length() != t.length()`, return `false` immediately.

Approach Using HashMap

1. **Count character frequencies of `s`**
   - Store occurrences of each character in a `HashMap`.
   - Use `big.put(a, big.getOrDefault(a, 0) + 1);` to update counts.
2. **Check characters of `t`**
   - Decrement the count in `HashMap` for each character in `t`.
   - If a character is missing, return `false`.
   - If a character count reaches **zero**, remove it.
3. **Return true if all characters matched correctly.**

Java Code

```
import java.util.HashMap;

class Solution {
    public boolean isAnagram(String s, String t) {
        if (s.length() != t.length()) return false; // Different lengths → not an an
agram

        HashMap<Character, Integer> big = new HashMap<>();
```

```
        // Store frequency of characters in 's'
        for (char a : s.toCharArray()) {
            big.put(a, big.getOrDefault(a, 0) + 1);
        }

        // Check characters of 't'
        for (char c : t.toCharArray()) {
            if (!big.containsKey(c)) return false;
            big.put(c, big.get(c) - 1);
            if (big.get(c) == 0) big.remove(c);
        }

        return true;
    }
}
```

Time Complexity

- **Building the HashMap:** `O(n)`
- **Checking** `t`**:** `O(n)`
- **Total Complexity:** `O(n)`, where `n` is the length of `s` (or `t`).

Common Mistakes & Fixes

1. **Using** `s[i]` **instead of** `s.charAt(i)`
    - Strings in Java are not arrays; use `charAt(i)` to access characters.
2. **Not handling character removal**
    - If a character count becomes `0`, remove it from the map to avoid unnecessary lookups.
3. **Using** `HashMap<String, Integer>` **instead of** `HashMap<Character, Integer>`
    - Store **characters**, not strings as keys in the `HashMap`.

Alternative Approach: Sorting

- Sort both strings and check if they are equal.
- **Time Complexity:** `O(n log n)` due to sorting.

```
import java.util.Arrays;

class Solution {
    public boolean isAnagram(String s, String t) {
        if (s.length() != t.length()) return false;
        return Arrays.equals(s.chars().sorted().toArray(), t.chars().sorted().toA
rray());
    }
}
```

Summary

✅ **Best Approach:** HashMap (`O(n)`)

✅ **Alternative Approach:** Sorting (`O(n log n)`)

✅ **Common Mistakes:** Using incorrect data types, not handling removals properly.