0002\_Easy\_ValidAnagram\_#242\_Breakdown

Problem:

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

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Examples:

**Example 1:**

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

**Output:** true

**Example 2:**

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

**Output:** false

Observations:

* The strings are only letter characters.
* The strings must be of equal size to be anagrams.

Questions:

* Can the strings be modified?
* Can new strings be created?
* Is extra space allowed?

What needs to be true to solve this problem:

* Each string needs to contain the same amount of each character.

How would a person solve this problem:

* Read through the first string, keeping a count of each unique character, ex. There are 3 of a, 2 of b, 4 of z, etc. Read through the second string, keeping a count of each unique character. Compare the count from each string. If the count is the same, the strings are anagrams.
* Rearrange each string alphabetically and compare the two strings.
* Try rearranging the second string into different permutations and comparing it to the first string.

Brute force:

* Rearrange each string alphabetically and compare the two strings.
* Time Complexity: O(nlogn), the highest time complexity is the sort algorithm.
  + Specifically: O(2nlogn + n), each string is sorted, then the strings are simultaneously iterated through to compare each element.
* Space Complexity: O(1), depending on the sort algorithm.

Optimization (BUD, bottlenecks, unnecessary code, duplicate code):

* If the strings are of different lengths, they cannot be anagrams.
* Instead of sorting the strings, an array can be used as “buckets” to keep track of how many of each character is used.
  + An array can be used if the number unique characters is known, ex. if only letters then an array of size 26 is used, if however the number of unique characters is not known then a HashMap needs to be used. The time and space complexity of remain the same.
* Instead of iterating through each string individually, they can be iterated through simultaneously, as the first string is iterated through it adds to the buckets and as the second is iterated through it subtracts from the buckets.
* The buckets can then be checked.
* Time Complexity: O(n), iterating through the string occurs once.
  + Specifically: O(n + c), n being the length of the string which is variable, and c being the size of the bucket array which is constant, (though it can be either 26 for alphabetical or more if numbers or Unicode characters are included).
* Space Complexity: O(1), the bucket array is a constant, so the extra space is negligible.
  + Specifically: O( c ), the size of the bucket array, which is 26 if only using only letters.

Pseudocode:

* Check if the strings are equal in size.
  + If the strings are not equal in size, return false, they cannot be anagrams.
* Create a bucket array to count the characters in each string.
* Iterate through both strings simultaneously.
  + As the first string is passed through, add its characters to the bucket array.
  + As the second string is passed through, decrement its characters from the bucket array.
* Iterate through the bucket array.
  + If a bucket has a value in it, return false, there exists a number of characters different in one of the string.
* Return true if the bucket array has all zeroes, the strings are anagrams.