Check Permutation

Problem Description: Given two strings, check if they are permutations of each other. Return true or false.

Sample Input:

```
abac
aacb
```

Sample Output:

```
true
```

How to approach?

The first thing to check as soon as we start the function will be the lengths of the two strings. If the two strings have different lengths, we can straight away return false. Now, let's discuss some approaches to this problem.

1. The brute force approach to this problem will be to generate all permutations of the first string and match them with the second string one by one. If any of the permutations match, we can return true otherwise we'll return false.
   The time complexity of this approach will be O(n!) which is extremely bad!

2. Another way to approach this problem will be to sort both strings and then compare them. If they are equal, we can return true, else we'll return false.
   The time complexity of this approach will be O(n * log(n)) which is much better than the complexity of the brute force approach but still has room for improvement

3. Another approach that doesn't involve sorting the strings exists. Basically, all we have to do is to compare the frequencies of the characters in both strings. If the frequencies match, then we can conclude both strings are permutations of each other, otherwise they are not.

Let’s discuss the implementation of approach #3.
We’ll maintain a frequency array of 256 size (256 being the number of characters that have an ASCII code) and initialize all elements of this array to 0. `frequency[str[i]]` will store the frequency of the character in `str` that has an ASCII code of `(int)str[i]`. Then we’ll iterate through both strings and update the frequency array. This update process will be a little different.
For the first string, we’ll increase the frequency of current character by 1 and for the second string, we’ll decrease the frequency of current character by 1.
This will result in the frequency array have all 0s if the two strings are permutations of each other. Otherwise, at least one element in the frequency array will be non-zero.

The pseudo-code for this approach is shown on the next page.

```
function checkPermutation(str1, str2):

  if(str1.length != str2.length):
    return false

  frequency[256] <- new array of integers initialised with 0

  for char in str1:
    asciiCode <- (int)char
    frequency[asciiCode] <- frequency[asciiCode] + 1

  for char in str2:
    asciiCode <- (int)char
    frequency[asciiCode] <- frequency[asciiCode] - 1

  for element in frequency:
    if(element != 0):
      return false

  return true
```