Permutations in a String

Question:
https://leetcode.com/problems/permutation-in-string/

Given two strings `s1` and `s2`, return `true` if `s2` contains a permutation of `s1`, or `false` otherwise.

In other words, return `true` if one of `s1`'s permutations is the substring of `s2`.

Example 1:

```
Input: s1 = "ab", s2 = "eidbaooo"
Output: true
Explanation: s2 contains one permutation of s1 ("ba").
```

Example 2:

```
Input: s1 = "ab", s2 = "eidboaoo"
Output: false
```

Constraints:

- `1 <= s1.length, s2.length <= 104`
- `s1` and `s2` consist of lowercase English letters.

Approach:

The first thing we require is to find the permutations.
I found a trick that didn't require calculating the permutation. Basically, we check if a string has the same characters as our original. If this condition is satisfied, we can say that it is a permutation of the original string.
So I used 2 arrays: One to calculate the count of each character in s1 and the other to calculate if the same count of characters is present in s2 or not.
NOTE: We can also do this using **hashmaps by mapping each character with its number of occurrences**.

Solution:

```
def checkInclusion(self, s1: str, s2: str) -> bool:
        l1 = [0]*26
        l2 = [0]*26
        for x in s1:
                l1[ord(x) - ord('a')] += 1
        for i in range(len(s2)):
                print(l1,l2)
                l2[ord(s2[i]) - ord('a')] += 1
                if i >= len(s1):
                        l2[ord(s2[i-len(s1)]) - ord('a')] -= 1
                if l1 == l2:
                        return True
        return False
```