38. Check If a String Can Break Another String Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or vice-versa. In other words s2 can break s1 or vice-versa. A string x can break string y (both of size n) if x[i] >= y[i] (in alphabetical order) for all i between 0 and n-1. Example 1: Input: s1 = "abc", s2 = "xya" Output: true Explanation: "ayx" is a permutation of s2="xya" which can break to string "abc" which is a permutation of s1="abc". Example 2: Input: s1 = "abe", s2 = "acd" Output: false Explanation: All permutations for s1="abe" are: "abe", "aeb", "bae", "bea", "eab" and "eba" and all permutation for s2="acd" are: "acd", "adc", "cad", "cda", "dac" and "dca". However, there is not any permutation from s1 which can break some permutation from s2 and vice-versa. Example 3: Input: s1 = "leetcodee", s2 = "interview" Output: true

PROGRAM:

```
def checkIfCanBreak(s1, s2):
    s1_sorted = sorted(s1)
    s2_sorted = sorted(s2)

    can_break_s2 = True
    for i in range(len(s1)):
        if s1_sorted[i] < s2_sorted[i]:
            can_break_s2 = False
            break

    can_break_s1 = True
    for i in range(len(s2)):
        if s2_sorted[i] < s1_sorted[i]:
            can_break_s1 = False
            break

    return can_break_s1 or can_break_s2
s1 = "abc"
s2 = "xya"
print(checkIfCanBreak(s1, s2))

s1 = "abe"
s2 = "acd"
```

```
print(checkIfCanBreak(s1, s2))


s1 = "leetcodee"
s2 = "interview"
print(checkIfCanBreak(s1, s2))
```

OUTPUT:

```
True
False
True
```

TIME COMPLEXITY: O(n log n)