Frequency Counters and Multiple Pointers

Download code

In this exercise, you'll utilize problem solving patterns to solve the following code challenges:

Problems

constructNote

Write a function called ***constructNote***, which accepts two strings, a message and some letters. The function should return ***true*** if the message can be built with the letters that you are given; otherwise, it should return ***false***.

Assume that there are only lowercase letters and no space or special characters in both the message and the letters.

Constraints:

Time Complexity: O(M + N) - If M is the length of message and N is the length of letters:

Examples:

```
constructNote('aa', 'abc') // false
constructNote('abc', 'dcba') // true
constructNote('aabbcc', 'bcabcaddff') // true
```

averagePair

Write a function called ***averagePair***. Given a sorted array of integers and a target average, determine if there is a pair of values in the array where the average of the pair equals the target average. There may be more than one pair that matches the average target.

Constraints:

Time Complexity: O(N)

Examples:

```
averagePair([1, 2, 3], 2.5); // true
averagePair([1, 3, 3, 5, 6, 7, 10, 12, 19], 8); // true
averagePair([-1, 0, 3, 4, 5, 6], 4.1); // false
averagePair([], 4); // false
```

twoArrayObject

Write a function called ***twoArrayObject*** which accepts two arrays of varying lengths.The first array consists of keys and the second one consists of values. Your function should return an object created from the keys and values. If there are not enough values, the rest of keys should have a value of null. If there not enough keys, just ignore the rest of values.

Examples:

```
twoArrayObject(['a', 'b', 'c', 'd'], [1, 2, 3]) // {'a': 1, 'b': 2, 'c': 3, 'd': null}
twoArrayObject(['a', 'b', 'c'], [1, 2, 3, 4]) // {'a': 1, 'b': 2, 'c': 3}
twoArrayObject(['x', 'y', 'z'], [1, 2]) // {'x': 1, 'y': 2, 'z': null}
```

sameFrequency

Write a function called ***sameFrequency***. Given two positive integers, find out if the two numbers have the same frequency of digits.

Examples:

```
sameFrequency(182,281) // true
sameFrequency(34,14) // false
sameFrequency(3589578, 5879385) // true
sameFrequency(22,222) // false
```

Constraints

Time Complexity - O(N + M)

separatePositive

Write a function called ***separatePositive*** which accepts an array of non-zero integers. Separate the positive integers to the left and the negative integers to the right. The positive numbers and negative numbers need not be in sorted order. The problem should be done in place (in other words, do not build a copy of the input array).

Examples:

```
separatePositive([2, -1, -3, 6, -8, 10]) // [2, 10, 6, -3, -1, -8]
separatePositive([5, 10, -15, 20, 25]) // [5, 10, 25, 20, -15]
separatePositive([-5, 5]) // [5, -5]
separatePositive([1, 2, 3]) // [1, 2, 3]
```

Constraints

Time Complexity: O(N)

isSubsequence

Write a function called ***isSubsequence*** which takes in two strings and checks whether the characters in the first string form a subsequence of the characters in the second string. In other words, the function should check whether the characters in the first string appear somewhere in the second string, without their order changing.

Examples:

```
isSubsequence('hello', 'hello world'); // true
isSubsequence('sing', 'sting'); // true
isSubsequence('abc', 'abracadabra'); // true
isSubsequence('abc', 'acb'); // false (order matters)
```

Constraints:

Time Complexity - O(N + M)

countPairs

Given an array of integers, and a number, find the number of pairs of integers in the array whose sum is equal to the second parameter. You can assume that there will be no duplicate values in the array.

Examples:

```
countPairs([3,1,5,4,2], 6) // 2 (1,5 and 2,4)
countPairs([10,4,8,2,6,0], 10) // 3 (2,8, 4,6, 10,0)
countPairs([4,6,2,7], 10) // 1 (4,6)
countPairs([1,2,3,4,5], 10) // 0
countPairs([1,2,3,4,5], -3) // 0
countPairs([0,-4],-4) // 1
countPairs([1,2,3,0,-1,-2],0) // 2
```

Constraints

Time Complexity - O(N * log(N))

or

Time Complexity - O(N)

Further Study

longestFall

Write a function called ***longestFall***, which accepts an array of integers, and returns the length of the longest consecutive decrease of integers.

Examples:

```
longestFall([5, 3, 1, 3, 0]) // 3, 5-3-1 is the longest consecutive sequence of decreasing integers
longestFall([2, 2, 1]) // 2, 2-1 is the longest consecutive sequence of decreasing integers
longestFall([2, 2, 2]) // 1, 1, 2 is the longest consecutive sequence of decreasing integers
longestFall([5, 4, 4, 4, 3, 2]) // 3, 4-3-2 is the longest
longestFall([9, 8, 7, 6, 5, 6, 4, 2, 1]) // 5, 9-8-7-6-5 is the longest
longestFall([]) // 0
```

pivotIndex

Write a function called ***pivotIndex*** which accepts an array of integers, and returns the pivot index where the sum of the items to the left equal to the sum of the items to the right. If there are more than one valid pivot index, return the smallest value.

Examples:

```
pivotIndex([1,2,1,6,3,1]) // 3
pivotIndex([5,2,7]) // -1  no valid pivot index
pivotIndex([-1,3,-3,2]) // 1 valid pivot at 2: -1 + 3 = 2 however there is a smaller valid pivot at 1: -1 = -3 + 2
```

Constraints

Time Complexity: O(N)

Solution

View our solutions