Frequency Counter Problems!

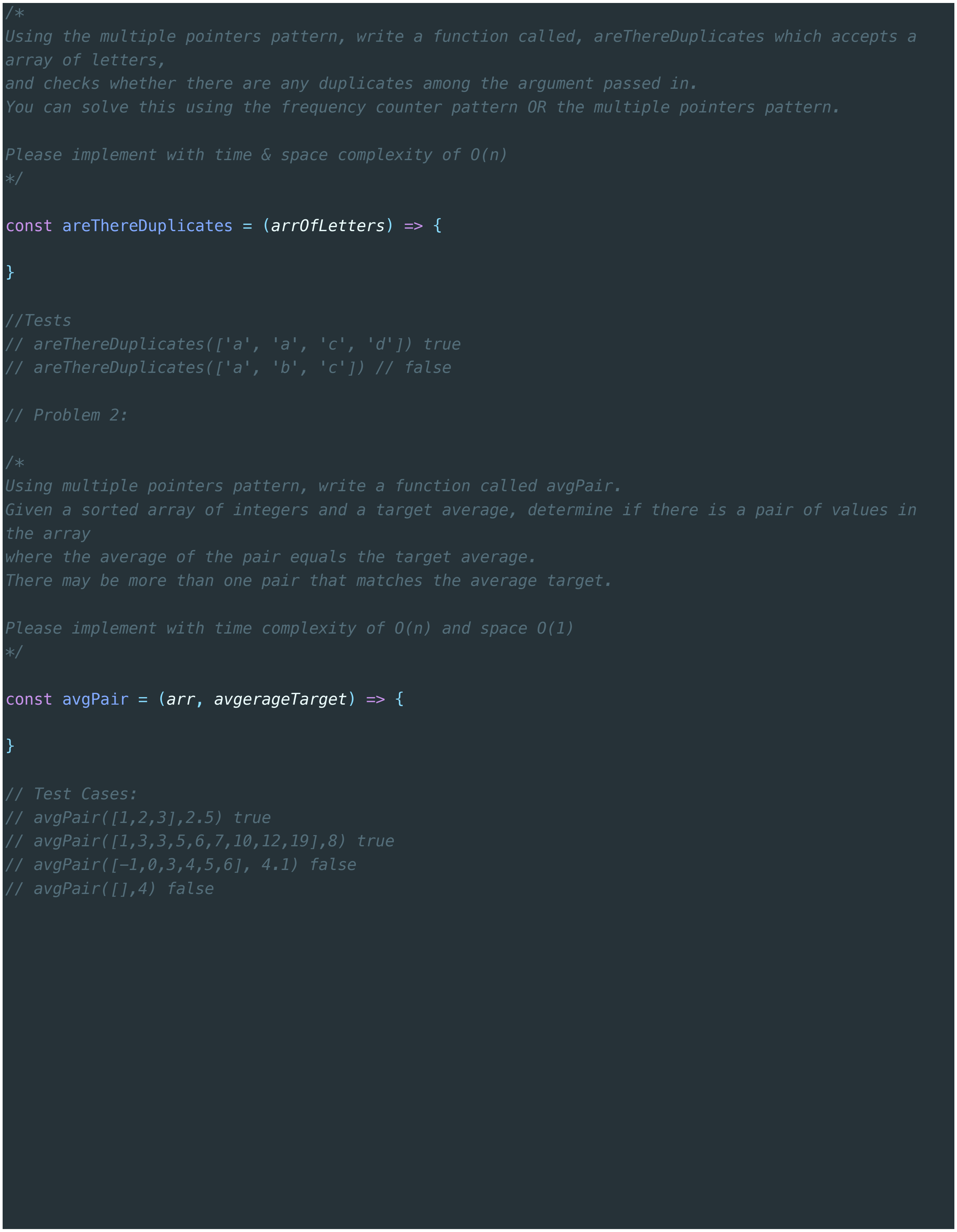
Problem #1: Frequency Counter Problem

```
/*
Using the frequency counter pattern, write a function called sameOccurences. Given two positive
integers,
find out if the two numbers have the same frequency of digits.

Write your algorithm in linear time complexity O(n)
*/

const sameOccurences = (num1, num2) => {

}

// Test Cases:
sameOccurences(123, 321) // true
sameOccurences(222, 22) //false
sameOccurences(83882, 23888) //false
```

Problem #2: Frequency Counter Pattern

```
/*
Given two strings, write a function to determine if the second string is an anagram of the first.
An anagram is a word, phrase, or name formed by rearranging the letters of another,
such as cinema, formed from iceman.
*/

validAnagram('', '') // true
validAnagram('aaz', 'zza') // false
validAnagram('anagram', 'nagaram') // true
validAnagram("rat","car") // false) // false
validAnagram('awesome', 'awesom') // false
validAnagram('qwerty', 'qeywrt') // true
validAnagram('texttwisttime', 'timetwisttext') // true

// {a: 0, n: 0, g: 0, r: 0, m: 0,s:1}
validAnagram('anagrams', 'nagaramm')

const validAnagram(first, second) => {

}
```

Please solve using Multiple Pointers Pattern



```
/*
Using the multiple pointers pattern, write a function called, areThereDuplicates which accepts a
array of letters,
and checks whether there are any duplicates among the argument passed in.
You can solve this using the frequency counter pattern OR the multiple pointers pattern.

Please implement with time & space complexity of O(n)
*/

const areThereDuplicates = (arrOfLetters) => {

}

//Tests
// areThereDuplicates(['a', 'a', 'c', 'd']) true
// areThereDuplicates(['a', 'b', 'c']) // false

// Problem 2:

/*
Using multiple pointers pattern, write a function called avgPair.
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.

Please implement with time complexity of O(n) and space O(1)
*/

const avgPair = (arr, avgerageTarget) => {

}

// Test Cases:
// avgPair([1,2,3],2.5) true
// avgPair([1,3,3,5,6,7,10,12,19],8) true
// avgPair([-1,0,3,4,5,6], 4.1) false
// avgPair([],4) false
// Problem 3:

/*
Write a function called subsequence 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.

Write your solution with time complexity O(n) and space O(1)
*/

const subsequence = (str1, str2) => {

}

// Test Cases:

// subsequence('hello', 'hello world') true
// subsequence('sing', 'sting') true
// subsequence('abc', 'abracadabra')  true
// subsequence('abc', 'acb') false
```