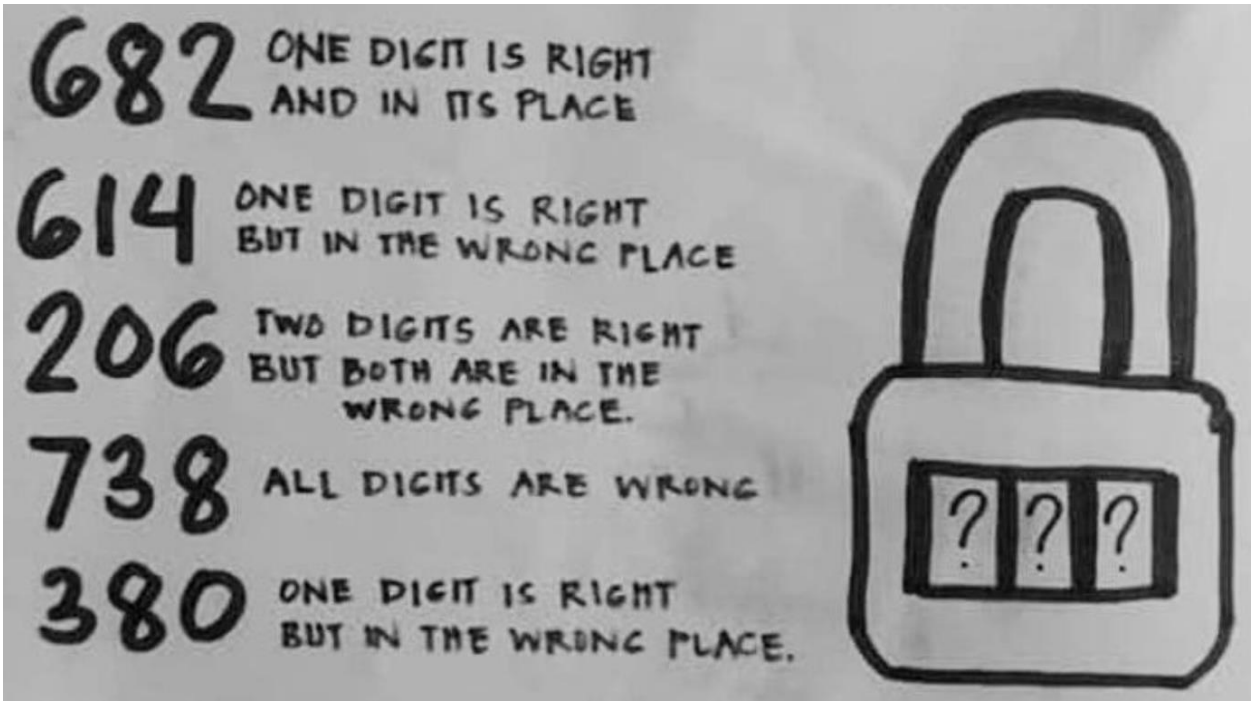
Please attempt this challenge and submit even if you only have a partial answer, we are keen to see your approach.

Write a code in a programming language of your choice to solve the following puzzle.

Sample Puzzle 2



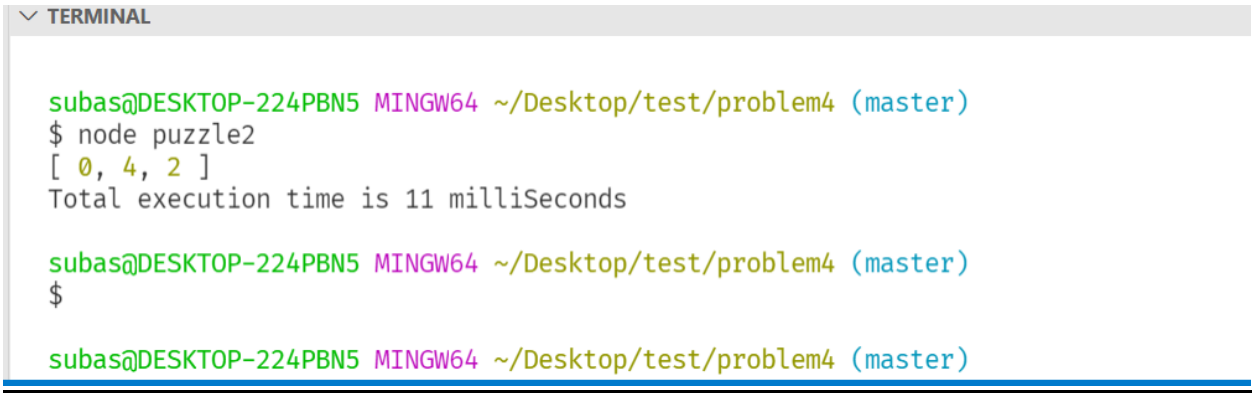
Source Code

```
let start = Date.now();

function totalCombinations(start, end) {
    const allCombinations = [];
    for (let i = start; i <= end; i++) {
        for (let j = start; j <= end; j++) {
            for (let k = start; k <= end; k++) {
                if ((i !== j) && (i !== k) && (j !== k) &&
                    (i !== 5) && (i !== 9) && (j !== 5) &&
                    (j !== 9) && (k !== 5) && (k !== 9)
                ){
                    allCombinations.push([i, j, k])
                }
            }
        }
    }
    return allCombinations;
}

function OneDigitRightButWrongPlace(checkValue, allCombinations, numberO
fTestDigits) {
    return allCombinations.filter(combination => {
        const numberOfMatches = [];
        const numberOfEqualPositions = [];
        // Find matching items
        checkValue.forEach(value => {
            if (combination.includes(value)) {
                numberOfMatches.push(value)
            }
        });
        // Find matching position
        for (let i = 0; i < checkValue.length; i++) {
            if (combination[i] === checkValue[i]) {
                numberOfEqualPositions.push(checkValue[i]);
            }
        }
        // Check if matching items is equal to number of test digits
        const numberOfOneMatches = numberOfMatches.length === numberOfTe
stDigits;

        // Check if number of matching position is 0
        const oneDifferentPosition = numberOfEqualPositions.length === 0
;
```

```
        return (numberOfOneMatches && oneDifferentPosition);
    })
}

function OneDigitRightAndRightPlace(checkValue, allCombinations, numberO
fTestDigits) {
    return allCombinations.filter(combination => {
        const numberOfMatches = [];
        const numberOfEqualPositions = [];
        // Find matching items
        checkValue.forEach(value => {
            if (combination.includes(value)) {
                numberOfMatches.push(value)
            }
        });
        // Find matching position
        for (let i = 0; i < checkValue.length; i++) {
            if (combination[i] === checkValue[i]) {
                numberOfEqualPositions.push(checkValue[i]);
            }
        }
        // Check if matching items is equal to number of test digits
        const numberOfOneMatches = numberOfMatches.length === numberOfTe
stDigits;

        // Check if number of matching position is equal to numberOfTest
Digits
        const oneEqualPosition = numberOfEqualPositions.length === numbe
rOfTestDigits;
        return (numberOfOneMatches && oneEqualPosition);
    })
}
function TwoDigitsCorrectButWrongPlace(checkValue, allCombinations, numb
erOfTestDigits) {
    return allCombinations.filter(combination => {
        const numberOfMatches = [];
        const numberOfEqualPositions = [];
        // Find matching items
        checkValue.forEach(value => {
            if (combination.includes(value)) {
                numberOfMatches.push(value)
            }
        });
        // Find matching position
        for (let i = 0; i < checkValue.length; i++) {
```

```
            if (combination[i] === checkValue[i]) {
                numberOfEqualPositions.push(checkValue[i]);
            }
        }
        // Check if matching items is equal to number of test digits
        const numberOfOneMatches = numberOfMatches.length === numberOfTe
stDigits;

        // Check if number of matching position is 0
        const oneEqualPosition = numberOfEqualPositions.length === 0;
        return (numberOfOneMatches && oneEqualPosition);
    })
}

function AllDigitAreWrong(checkValue, allCombinations) {
    return allCombinations.filter(combination => {
        let doesMatch = false;
        for (i = 0; i < checkValue.length; i++) {
            if (combination.includes(checkValue[i])) {
                doesMatch = true;
            }
        }
        return !doesMatch;
    })
}

const allCombinations = totalCombinations(0, 9)

const filter_1 = AllDigitAreWrong([7, 3, 8], allCombinations);

const filter_2 = OneDigitRightAndRightPlace([6, 8, 2], filter_1, 1);

const filter_3 = OneDigitRightButWrongPlace([6, 1, 4], filter_2, 1);

const filter_4 = TwoDigitsCorrectButWrongPlace([2, 0, 6], filter_3, 2);

const filter_5 = OneDigitRightButWrongPlace([3, 8, 0], filter_4, 1);

result = filter_5[0];
console.log(result)

let end = Date.now();
const totalExecTime = end - start;
console.log(`Total execution time is ${totalExecTime} milliSeconds`);
```

Output



Summary

Process is similar to puzzle 1 as done earlier. It can be made possible using functional programming.