HW1: Pseudocode and Recursion

Instructions

The goal of this homework is to practice what we covered in Lecture 2 and 3. Please review both lectures in order to have the context to solve these problems.

Submitting This Assignment

iCollege

In order to complete the coding portions of this assignment, you must make commits of your code. We will take the latest commit as your "final submission".

Grading and Corrections

We've provided some example inputs and outputs to help you test your code and check your work. Producing the correct output for these examples is necessary but not sufficient for receiving full credit. There may be other test cases that we use in grading--you should consider writing your own examples.

Java Language API

You may use more basic data types like int[ ], float [ ], matrix like [ ][ ]. For Java Abstract ones like ArrayList<>, List<>, it's better for your mastering of Java knowledge. Otherwise, you can just ignore them (or simple convert [ ] to ArrayList at the end).

Focus on Algorithms' details, not Java Language. Runnable, correctness, efficiency. (well organized, easy to read, not in a mass)

Showing Your Work

1. You should add comments that explain the key ideas behind your approach.
2. You may also add additional test cases in order to ensure that your code is 100% correct.

Q1: Find Missing Number (2 points)

Convert the following pseudocode algorithm into Java. You'll find example tests within the main method to ensure you're headed in the right direction. **Make sure you understand each example.**

```
algorithm findMissing
  Input: integer array A of length N where each element is distinct
    and in the range [0, N]
  Output: integer x where x is in the range [0, N], but not in A

  s = the sum of all numbers in A
  return (N(N+1))/2 - s
```

Q2: TwoSum (Fast) (3 points)

Convert the following code into Pseudocode and put it in twosum.txt. **Hint:** for *some* loops, you *should* describe what it does in English. **There are no tests for this problem, so be sure to double check and test your translation manually, similar to how we did in lecture.**

```
 public static int[] twoSumFast(int[] arr, int target) {
    HashSet<Integer> seen = new HashSet<>();
    for (int j = 0; j < arr.length; j++) {
      int otherAddend = target - arr[j];
      if (seen.contains(otherAddend)) {
        for (int i = 0; i < arr.length; i++) {
          if (arr[i] == otherAddend)  {
            return new int[] {i, j};
          }
        }
      } else {
        seen.add(arr[j]);
      }
    }
    return new int[] {-1, -1};
  }
```

Q3: countFives (2 points)

Write countFives, which takes in an integer and returns the number of times 5 appears as a digit within the number. Examples:

```
  countFives(123467890) // should output 0
  countFives(555555)    // should output 6
  countFives(15354)     // should output 2
```

In order to receive full credit for this problem, you must use recursion. I.e. using =, for, while, etc. is prohibited.

Hint: recall the % and / operators:

```
123 % 10 // evaluates to 3
123 / 10 // evaluates to 12
```

Q4: pickTrees (3 points)

You build homes out of wood and you need material from a nearby forest. However, you want to avoid deforestation, so you decide for each tree you cut down, you'll leave its neighbors alone, giving the forest time to recover. However, you still need as much wood as possible, so you have to be careful about which trees you pick to cut down.

Write pickTrees, which takes in an array of N trees arr where arr[i] represents how much wood you can harvest by cutting down tree i. It should return the max amount of wood you can harvest while following the rule of skipping neighbors:

```
// Pick tree 0, tree 2, and tree 4 => 1 + 3 + 5 = 9 wood total
int testResult5 = pickTrees(new int[] {1, 2, 3, 4, 5});
System.out.println(testResult5); // should output 9

// Pick tree 1 and tree 3 => 3 + 3 = 6 wood total
int testResult6 = pickTrees(new int[] {1, 3, 4, 3});
System.out.println(testResult6); // should output 6

// Pick tree 0 and tree 3 => 5 + 9 = 14 wood total
int testResult7 = pickTrees(new int[] {5, 1, 4, 9});
System.out.println(testResult7); // should output 14
```

In order to receive full credit for this problem, you must use recursion. I.e. using =, for, while, etc. is prohibited.