Introduction to Programming

Labs – Week 9b

Note: Only Exercise 1 & 2 need to be solved using recursion.

Exercise 1

Write a recursive function that takes in one argument `n` and computes `n!`, the factorial function. Recall that `n! = n*(n-1)!` when `n>0` and `0! = 1`.

Exercise 2

Write a recursive function `count7()` that given a non-negative integer `n`, returns the count of the occurrences of `7` as a digit, so for example `717` yields `2`. Following are some examples

```
count7(7170123) → 2
count7(7)       → 1
count7(123)     → 0
```

Hint: Note that mod (`%`) by `10` yields the rightmost digit (`126%10` is `6`), while divide (`/`) by `10` removes the rightmost digit (`126/10` is `12`).

Exercise 3

We'll say that an element in an array is *alone* if there are values before and after it, and those values are different from it. Write a function `notAlone()` that return a version of the given array where every instance of the given value which is alone is replaced by whichever value to its left or right is larger.

```
notAlone({1, 2, 3}, 2) → {1, 3, 3}
notAlone({1, 2, 3, 2, 5, 2}, 2) → {1, 3, 3, 5, 5, 2}
notAlone({3, 4}, 3) → {3, 4}
```

Exercise 4

Write a function `canBalance()` that given a non-empty array, return `true` if there is a place to split the array so that the sum of the numbers on one side is equal to the sum of the numbers on the other side.

```
canBalance({1, 1, 1, 2, 1}) → true
canBalance({2, 1, 1, 2, 1}) → false
canBalance({10, 10}) → true
```

Exercise 5

Write a function `scoresAverage()` that given an array of scores, compute the `int` average of the first half and the second half, and return whichever is larger. We'll say that the second half begins at index `length/2`. The array length will be at least `2`. To practice decomposition, write a separate helper method `int average(int[] scores, int start, int end)` which computes the average of the elements between indexes `start..end`. Call your helper method twice to implement `scoresAverage()`. Normally you would compute averages with `double`s, but here we use `int`s so the expected results are exact.

```
scoresAverage({2, 2, 4, 4}) → 4
scoresAverage({4, 4, 4, 2, 2, 2}) → 4
scoresAverage({3, 4, 5, 1, 2, 3}) → 4
```

Exercise 6

Write a program that randomly fills in `0`s and `1`s into a 4-by-4 matrix, prints the matrix, and finds the first row and column with the most `1`s. Here is a sample run of the program:

```
0011
0011
1101
1010

The largest row index: 2
The largest column index: 2
```

Implement and use following functions in your program:

```
public static int[][] genRandomMatrix(int m, int n)
public static void printMatrix(int[][] a)
public static int findMaxOnesRow(int[][] a)
public static int findMaxOnesCol(int[][] a)
```

Exercise 7

Nine coins are placed in a 3-by-3 matrix with some face up and some face down. You can represent the state of the coins using a 3-by-3 matrix with values 0 (heads) and 1 (tails). Here are some examples:

```
0 0 0      1 0 1      1 1 0      1 0 1      1 0 0
0 1 0      0 0 1      1 0 0      1 1 0      1 1 1
0 0 0      1 0 0      0 0 1      1 0 0      1 1 0
```

Each state can also be represented using a binary number. For example, the preceding matrices correspond to the numbers

```
000010000  101001100  110100001  101110100  100111110
```

There are a total of 512 possibilities, so you can use decimal numbers 0, 1, 2, 3, . . . , 511 to represent all states of the matrix. Write a program that prompts the user to enter a number between 0 and 511 and displays the corresponding matrix with the characters H and T. In the following sample run, the user entered 7, which corresponds to 000000111. Since 0 stands for H and 1 for T, the output is correct.

```
Enter a number between 0 and 511: 7

H H H
H H H
T T T
```

Implement and use following functions in your program:

```
Public static int[][] toMatrix(int x)
public static void printCoinsMatrix(int[][] a)
```