NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

Practical Assessment for Semester 1, AY2020/21
CS1010J — Programming Methodology

31 October 2020 Time Allowed: 1.5 hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of 2 exercises on 7 pages.
2. This is an **OPEN BOOK** assessment. You may refer to print copy or digital version of your module materials (e.g. lecture notes, tutorial paper) stored on your iPad, laptop, etc.
3. Calculators are allowed.
4. In line with the university rules, any form of communication with other students, or the use of unauthorised materials is considered cheating and you are liable to the disciplinary action.
5. The first 10 minutes of the assessment is reserved for you to read questions and design algorithms. You are not allowed to type your programs in this period of time.
6. Your invigilator will send you the question paper and skeleton programs. You are to use **DrJava** to write your programs. No other IDEs are allowed.
7. You may assume that all input data are valid; hence no input validation is needed.
8. You are responsible for submitting your programs to CodeCrunch by the end of the assessment. **Do not leave your submission to the last minute in case of network congestion.** You have 99 chances of submissions and only the last submission will be graded.
9. You are **NOT** allowed to use any Java language syntax/knowledge not covered in weeks 1-9. If in doubt, please raise it in the practice assessment forum.
10. Grading of the practical assessment may take a week or more.

ALL THE BEST!

Example Rough Marking Scheme for an Exercise of 50 Marks

1. Style: 10 marks
   - Write program description and student number in the program header.
   - Write a short and meaningful description for every method (except main).
   - Apply consistent indentation and good naming of variables.
2. Correctness and design: 40 marks
   - Graders will manually go through your program and award marks method by method.
3. Additional penalties:
   - Program is not compilable: deduct 5 marks.
   - Break coding restrictions: deduct 5 - 20 marks, depending on severity.

Exercise 1: An Array of Integers [50 marks]

Given an array of distinct positive integers and another positive integer t, perform the following three tasks.

1. Report the largest difference between adjacent integers in the array.

   For example, if the array is {13, 3, 4, 21}, the largest difference between neighbours is 17 (= 21 - 4). Note that the difference should be reported as a positive integer.

2. Report the number of **tough numbers** in the array.

   A tough number is an integer num of the form $num = k * 2^n + 1$, where k and n are positive integers, k is odd, and $k < 2^n$.

   A few examples are shown below.

   - 3 is a tough number because 3 = 1*2^1 + 1 and 1 < 2^1.
   - 4 is NOT a tough number.
   - 13 is a tough number because 13 = 3*2^2 + 1 and 3 < 2^2.
   - 21 is NOT a tough number. It can be expressed as 21 = 5*2^2 + 1 but 5 > 2^2.

   Consequently, the array {13, 3, 4, 21} contains 2 tough numbers.

3. Report the number of **smiling numbers** in the array.

   A smiling number is a positive integer that has the sum of its digits equals to a given positive integer t.

   For example, if the array is {13, 3, 4, 21} and t = 4, there are 2 smiling numbers in the array: 13 and 4; the sum of digits of each of them is 4.

In the given skeleton program **ArrayOfInts.java**, complete the following 4 methods.

1) **`int maxDiff(int[] arr)`** that returns the largest difference between adjacent integers in **`arr`**.
2) **`boolean isToughNum(int num)`** that checks if **`num`** is a tough number. It returns true if so, or false otherwise.
3) **`int countToughNum(int[] arr)`** that returns the number of tough numbers in **`arr`**.
4) **`int countSmilingNum(int[] arr, int t)`** that returns the number of smiling numbers in the array, i.e. the number of array elements that have the sum of their digits equals to **`t`**.

You may assume that **`arr`** contains at least 2 elements. Moreover, it doesn't contain duplicate elements.

Take note of the following additional instructions/restrictions:

1. You are **not** allowed to change the **`main()`** method or method headers of other given methods.
2. Your methods may invoke each other and you may create additional arrays or write additional methods as necessary.

Two sample runs of the program are shown below with the user's input shown in **bold**.

Sample run #1

```
Enter the size of array: 4
Enter the array: 13 3 4 21
Maximum difference = 17
Number of tough numbers = 2
Enter t: 4
Number of smiling numbers = 2
```

Sample run #2

```
Enter the size of array: 5
Enter the array: 11 14 23 9 5
Maximum difference = 14
Number of tough numbers = 2
Enter t: 5
Number of smiling numbers = 3
```

Exercise 2: Employees [50 marks]

A company stores employee information in a table. One example is shown in Table 1 below. Each row of the table shows the information of one employee and it contains three fields: **staff ID**, **age** and **salary**. You may assume that all three fields are positive integers, staff ID is unique to each employee, and age is between 18 and 67, both ends inclusive.

ID	Age	Salary
111	35	4500
112	27	6600
113	25	2000
114	60	10000
115	60	6000
117	35	5000
118	60	30000
119	27	6600
122	40	4000
123	35	3400

Table 1. Records of 10 employees

Task 1

Suppose the above table is implemented as a two-dimensional (2D) array in your program. The company would like you to sort the 2D array by **age** in non-decreasing order, and for those employees with the same age, further sort by **salary** in non-decreasing order. If two employees have the same age and salary, their relative order in the original table should be preserved. Table 2 below shows the result of sorting Table 1 according to the stated criteria.

ID	Age	Salary
113	25	2000
112	27	6600
119	27	6600
123	35	3400
111	35	4500
117	35	5000
122	40	4000
115	60	6000
114	60	10000
118	60	30000

Table 2. Records of 10 employees after sorting

1) Write a method **`void swap(int[][] employees, int row1, int row2)`** that swaps two rows, **`row1`** and **`row2`**, of the **`employees`** array.

2) Write a method **`void sort(int[][] employees)`** that sorts the **`employees`** array according to the criteria stated on the previous page. You may choose selection sort, bubble sort or enhanced bubble sort covered in CS1010J. However, you are **not** allowed to call Java sorting APIs (e.g. you cannot use Arrays.sort()).

Task 2

The company is interested to know which age most employees are of (i.e. most common age).

3) Write a method **`int[][] mostCommonAge(int[][] sortedEmployees)`** that checks which age is most common among employees. Note that there may be more than one most common age (see example below). The method then stores respective employees in a new 2D array and returns this new array. Respective employees should be stored in the new 2D array in the same order they appear in the parameter **`sortedEmployees`** array. You may assume that the company has at least one employee. Note that **`sortedEmployees`** array is already sorted in Task 1.

For the example shown in Table 2 on the previous page, there are two most common ages: 35 and 60. Hence the **`mostCommonAge()`** method will return a new array containing the following six employees.

ID	Age	Salary
123	35	3400
111	35	4500
117	35	5000
115	60	6000
114	60	10000
118	60	30000

Table 3. Two ages with the most number of employees

Complete the skeleton program **Employees.java** for the above two tasks. Take note of the following additional instructions/restrictions:

1. **`main()`**, **`readInput()`** and **`printArray()`** methods are complete and given. You are **not** allowed to change them.
2. You are **not** allowed to change the method headers of other given methods.
3. Your methods may invoke each other and you may create additional arrays or write additional methods as necessary.

Two sample runs of the program are given below with the user's input shown in **bold**. For your convenience of testing, test inputs are also appended to the back of the skeleton program.

Sample run #1

```
Enter the number of rows: 4
Enter data:
114 60 10000
115 60 6000
117 35 5000
118 60 30000
Sorted by age, then by salary:
117 35   5000
115 60   6000
114 60   10000
118 60   30000
Employees with the most common age:
115 60   6000
114 60   10000
118 60   30000
```

(Sample run #2 on the next page)

Sample run #2

```
Enter the number of rows: 10
Enter data:
111 35 4500
112 27 6600
113 25 2000
114 60 10000
115 60 6000
117 35 5000
118 60 30000
119 27 6600
122 40 4000
123 35 3400
Sorted by age, then by salary:
113 25   2000
112 27   6600
119 27   6600
123 35   3400
111 35   4500
117 35   5000
122 40   4000
115 60   6000
114 60   10000
118 60   30000
Employees with the most common age:
123 35   3400
111 35   4500
117 35   5000
115 60   6000
114 60   10000
118 60   30000
```

=== END OF PAPER ===