Assignment overview

Student pranksters from the School of Business are Zoombombing our labs and department meetings! They have been joining our Zoom calls uninvited, talking over our students and instructors, playing loud music and other noise through their microphones, and otherwise just being pesky and disruptive. Computing instructors are growing tired of having to constantly mute/remove them from the meetings.

In response, Stephanie is going to start generating secure passcodes for everyone to use for Zoom calls. She needs to inform everyone whenever there is a new passcode, but if she simply emails it out, it will be intercepted by the interlopers, who will continue their meddlesome invasions.

Stephanie decides to obscure each new passcode using a clever method that we programmers can crack, while making it harder for the troublemakers. The message will consist of a long list of supposed passcodes, *in scrambled order*. The new passcode is the smallest number that DOES NOT appear somewhere in the list.

You must write a program to determine the new passcode. There are different ways this could be done, but here is the two-step process we will use:

1. Sort the list of passcodes from Stephanie's message
2. Determine the smallest passcode that is missing from the sorted list

You must solve these problems with Divide-and-Conquer and Decrease-and-Conquer algorithms, as specified in the sections below.

Given code

Find_the_Passcode.java contains some sample Java code that will read a file containing a list of integers and save it into an array. You may use it or not, as you wish. If you have your own favourite way to read an array of integers from a data file, that is fine.

Sample input files

For your testing purposes there are several sample input files available on Learning Hub. Each contains a different list of numbers and therefore each will give a different answer for the new passcode. You should test your program with all of these input files. The list of numbers in each file will include 0. (Unless it just so happens that 0 is the new passcode!)

Part 1 (4 marks) – Sort the array

Implement the MergeSort algorithm as a function that sorts an array of integers. The array should be the only argument to this function, and the function return type should be `void`. You may write additional helper functions as need.

Note: as part of MergeSort it is necessary to copy portions of one array to another. It is allowable to use the static methods Arrays.copyOf() or Arrays.copyOfRange() for this. (You may also perform this "by hand" with a for or while loop.)

Part 2 (4 marks) – Find the passcode

Implement a function that finds the smallest *missing* number from a *sorted* array of integers. The primary function should take only the array as an argument, and will return the smallest missing number (an integer). You may write helper functions as needed.

Full marks (4/4) – implement this as a "decrease by half" and conquer algorithm

Partial marks (2/4) – implement this as a "decrease by 1" and conquer algorithm

Hint: As inspiration/guidance for the "decrease by half" algorithm, think about binary search.

Part 3 (3 marks)

Include a main program that reads a data file, sorts the data (using your function for Part 1), finds the passcode (using your function for Part 2) and displays the new passcode that is determined by the data file. You may start with the code that is included on Learning Hub, but you don't have to.

For simplicity you should treat the passcodes as ordinary integers during processing (sorting the list, and finding the smallest missing value). But whenever you *output* a passcode, make sure it is the same length as all the other passcodes in the data file. For example, if the data file contains 4-digit passcodes, then the value 37 should be displayed as "0037". Here is one way that you can format integers with leading zeroes (the '4' determines the total length of the output integer):

```
System.out.println(String.format("%04d", passcode));
```

Part 4 (4 marks)

You must use good programming style throughout your code, including:

- Header comments with at least the following information:
    - Your name, ID, and set
    - A short description of the program (a sentence is enough)
- A descriptive comment for any significant block of code such as a function or an important loop
- Comments for any other code that is "unusual" or otherwise interesting
- Meaningful variable names for important variables

Tips and tricks

If you have any difficulty completing Part 1 (MergeSort), you can still work on Part 2 by temporarily using Arrays.sort() to sort your list of passcodes. Obviously there would be no credit for Part 1 if you submit your code this way, but the purpose is not to let problems with Part 1 prevent you from successfully completing Part 2.

More hints

The decrease-by-half algorithm for Part 2 should strongly resemble Binary Search. But there is an important difference: You are not looking for the index where a particular value is found – you are looking for a passcode that *is NOT found*.

Suppose this is your input array after sorting:

00 01 02 03 04 05 06 07 08 09 10 11 12 13 16 17 18 19 20 21 23

Then the correct new passcode should be 14. *How do you know when you've found it?*

Also note: You may need to examine not only the "middle" element but one or both of its neighbours. Be careful in this case to avoid array-out-of-bounds errors!

Submission and marking

Due date: As shown on Learning Hub. Your last submission will be the one that counts. Late assignments will not be graded.

Submit: Your Java source code (file name not important)

Marking: This lab is worth 15 marks.