Assignment overview

Student pranksters from the School of Business are invading our computer labs! They have been sneaking into our labs on the weekends, throwing accounting-and-pizza bacchanals, burning out our printers with reams of TPS reports, and filling our whiteboards with endless financial scribbles. And they've been leaving behind their messes for us to clean up on Monday mornings!

In response, Stephanie is going to generate new numeric passcodes for our labs every week. 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 very long, unsorted list of *incorrect passcodes* (all the same length). The correct new passcode is the smallest number that *does not appear* anywhere on the list. Note: The correct passcode could be any sequence of digits, including all zeroes. The length of the passcode is determined by the length of all the *fake* passcodes in the data file.

You must write a program to read a data file and determine the new passcode. As always there are many ways to do this, 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.

Please note that this program is using "plain arrays". No ArrayList please.

Sample input files

For your testing purposes there are several sample input files available. 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 may or may not include zero (depends on whether the new passcode just happens to be a sequence of all zeroes).

Part 1 (5 points) – 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. I recommend using the algorithm as presented in the lecture notes, but as long as it is MergeSort and has the same "signature" that's OK.

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 write code for this "by hand" with your own loop.

Part 2 (5 points) – Find the passcode

Implement a function that finds the smallest missing number from a sorted array of distinct integers. The 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 (5/5) – implement this as a "decrease by half" and conquer algorithm

Partial marks (2.5/5) – implement this as a "decrease by 1" and conquer algorithm

Hint: As inspiration/guidance for the "decrease by half" algorithm, think about binary search. (More hints below.)

Part 3 (5 points) – Main program

Your main program should:

- Read a data file
- Sort the list of (fake) passcodes (Part 1)
- Find the smallest missing value (the actual passcode) (Part 2)
- Output a message indicating the new passcode

For simplicity, treat the passcodes as ordinary integers during all processing (reading the data, sorting the list, and finding the smallest missing value). However, whenever you *output* any passcode, always include leading zeroes if needed so that the length of the passcode is consistent with all of the passcodes in the input data file. Different data files have different length passcodes.

For example, if the data file contains 4-digit passcodes, then the value 37 should be displayed as "0037". However, if the data file contains 5-digit passcodes, then this passcode would be "00037".

Here is one way that you can format integers with leading zeroes in Java (the '4' determines the total length of the output integer):

```
System.out.println(String.format("%04d", 37)); // output is 0037
```

Part 4 (5 points)

You must follow all the same coding style guidelines as outlined in our first coding assignment.

In addition, you should never output an unlabeled/unidentified/unexplained number.

Do not just output the passcode!

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 point here is not to let problems with Part 1 prevent you from successfully completing Part 2.

Are you having trouble testing your search algorithm because you don't know the correct answer for the data files?

- The file names appear to contain the correct answers – but you should really always verify that sort of thing independently if you can
- Think of a brute-force algorithm that you can write quickly for testing purpose and have confidence that it is correct
- Make up your own (possibly very small) data file for which you know the correct answer

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*.

For example, 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?* Draw this array with the indices showing and see if you can spot any clues.

Also note: You may need to examine not only the "middle" element of an array but also one or both of its neighbours at a given moment. Watch out for array-out-of-bounds errors!

How, what, and when to submit:

Please submit the following to the dropbox on Learning Hub:

- Just your Java source code
- File names not important

You may (and should!) discuss the lab and coding techniques with your classmates, but all of the work you submit to Learning Hub must be your own.

This lab is worth 20 points.

It is due at midnight next Wednesday.