DUE DATE: **By the end of your lab session**

CSE-271: Object-Oriented Programming

Exercise #9

Max Points: 20

Name: Raobot

For your own convenient reference – You should first save/rename this document using the naming convention `MUid_Exercise9.docx` (example: `raodm_Exercise9.docx`) prior to proceeding with this exercise.

Objectives: The objectives of this exercise are to:

1. Practice the use of recursion for problem solving
2. Review sorting algorithms
3. Experiment with measuring runtime of algorithms
4. Review analytical metrics for algorithms
5. Use concepts relating to searching & sorting for problem-solving

Fill in answers to all of the questions. For some of the questions you can simply copy-paste appropriate text from Eclipse output into this document. You may discuss the questions or seek help from your neighbor, TA, and/or your instructor.

Part #0: One time setup of Eclipse (IDE) – Only if needed

We already configured `Eclipse`'s source formatter and Checkstyle plug-in as part of Lab #1. If your `Eclipse` is not configured (because you are using a different computer) then use the instructions from Lab #1 to configure `Eclipse`.

Part #1: Recursion: Concepts & Tracing

Estimate time: < 20 minutes

Background (from Wikipedia): In computer science, recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem Recursion solves such recursive problems by using functions that call themselves from within their own code. The approach can be applied to many types of problems, and hence, recursion is one of the central ideas of computer science.

Recursion – noun. Definition – See recursion

Recursive acronym: A recursive acronym is an acronym where the first letter is the acronym itself. The acronym can be expanded to infinity. For example, **GNU** stands for "*GNU's Not Unix*."

Exercise: Briefly (2-to-3 sentences each) respond to the following questions regarding generic concepts of Recursion

1. What is a stack overflow (no, no, this is not asking about StackOverflow.com; but yes, this is the concept underlying the website's name)? When does it occur?

   Stack is a portion of memory that is used to store parameters and local variables for a method. Hence, each time a method is called, it uses some stack space. As more and more methods are called, more stack space is used until the program eventually runs out of stack space. This situation, where the program has used up all of its stack memory is called "Stack overflow".

   Stack overflow can occur any time there is a deep method-call hierarchy. This situation also commonly occurs in buggy recursive methods.

2. Given the following implementation for a `mystery` method, illustrate the output generated by the calls to the `mystery` method shown below:

```java
public static void mystery(int n) throws Exception{
  if (n < 0) {
    throw new Exception();
  }
  try {
    mystery(n - 1);
    System.out.println("*".repeat(n));  // Prints n '*'s
  } catch (Exception e) {
    System.out.println("Hello");
  }
}
```

   a. What is the output generated by the method call: `mystery(0)`?

```
Hello
*
**
***
****
```

   b. What is the output generated by the method call: `mystery(-2)`?

   Exception

   c. What is the output generated by the method call: `mystery(4)`?

```
Hello
*
**
***
****
```

Part #2: Sorting and time complexity

Estimated time: < 20 minutes

Background: In computer science, time complexity denoted by O(n) notation, is an important and valuable approach for comparing algorithms. The objective of the big-O notation is to provide an intuitive yet powerful approach to compare change in runtime of algorithms, as number of inputs *n*, increases. It is important to keep the following relationship in mind:

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

Exercise:

1. For each sorting description identify the sort that is best being described (Bubble Sort, Selection Sort, Merge Sort, Quick Sort):

/

   a. ________________________________: This algorithm for sorting finds the smallest item in the list and places it in the beginning of the list. It then finds the 2nd smallest item and places it at the second spot of the list.

/

   b. ________________________________: This algorithm of sorting partitions the list into two sub-lists, such that everything smaller than a specific pivot element is before the pivot and everything larger than the pivot is after it. It then sorts all items before the pivot. Finally, it sorts all items after the pivot.

2. Assume that you have the option to use Merge Sort or Selection Sort to sort a large list of numbers. Which one of the two sorting methods must be used and why (1 sentence)?

   The time complexity of Merge Sort is *O(n log n)* which is better than the time complexity of Insertion Sort which is *O(n^2)*. Consequently, Merge sort should be used.

3. For the following operations, indicate the expected time complexity. In addition, briefly describe how you deduced the time complexity.

Source code	Big-O	How was time complexity deduced?
`"abcdef".indexOf('c');`	O(n)	Because 'n' characters in the string have to be checked to find index or return -1.
`"12345".compareTo("12345");`	O(n)	Because 'n' characters have to be compared to determine if strings are equal.

`int[] list = {1, 2, 3, 4, 5};`
`Arrays.binarySearch(list, 5);`
	O(*log* n)	Because binary search is O(log n) algorithm.

`void insert(ArrayList<Integer> list) {`
`  int mid = list.size() / 2;`
`  list.add(mid, -1);`
`}`
	O(n)	Because inserting requires copying 'n' values 1-spot over to create space for value to be inserted.

`void modify(ArrayList<Integer> list) {`
`  int mid = list.size() / 2;`
`  list.set(mid, -1);`
`}`
	O(1)	Because changing an existing value in a list is a constant time operation.

`int[] list = {1, 2, 3, 4, 5};`
`Arrays.sort(list);`
	O(n *log* n)	Because the Javadoc documentation says quicksort is used and we know quicksort in a O(n log n) algorithm.

`Float get2nd(ArrayList<Float> list) {`
`  Collections.sort(list);`
`  return list.get(1);`
`}`
	O(n *log* n)	The step with highest time complexity is sort which is merge sort from Javadoc. We know merge sort is O(n log n).

4. Java has two different sorting methods, namely: `Arrays.sort` and `Collections.sort`. What is the difference between the two methods?

   There are two key differences between the above two JDK methods even though both are O(n *log* n) methods:
   - First, they work on different data types. `Array.sort` works on arrays of primitive types. The `Collections.sort` method operates on classes that implement Collection interface. E.g.: `ArrayList`, `Deque` (pronounced deck)
   - Second, based on the Javadoc, `Arrays.sort()` implements dual-pivot quicksort while `Collections.sort()` uses merge sort.

5. Suppose we have two programs, Program-α and Program-β that organize a list of objects in a similar manner such that the resulting lists from the two programs are identical. Further, suppose that Program-α takes *O(n)* steps and Program-β takes *O(log n)* steps. Based on this information which one of the two programs should be used?

   Since *O(log n)* is better than *O(n)*, Program-β should be used.

Part #3: Experimental runtime analysis

Estimated time: < 20 minutes

Background: Runtime analysis of programs is often conducted by measuring the time it takes for the program to run. This approach is referred to by different names, including: ❶ benchmarking, ❷ profiling, ❸ empirical runtime analysis, and ❹ experimental analysis. There are several different approaches to performing experimental analysis. A simple approach is to measure elapsed time when a method call is made with different sizes of input and plot the data to analyze runtime behaviors.

Concept check:

1. State one key advantage of benchmarking (aka experimental analysis)

   The advantages of experimental analysis include:

- Provides an actual performance data and resource consumption
- Only way to determine and optimize runtime-constants
- Directly enables planning/procuring for hardware resources etc.
- Ultimately, this is what people/customers care about

2. State one disadvantage of profiling (aka experimental analysis)

Some of the disadvantages of experimental analysis are:
- Requires development, which can be time consuming
- Specific to the platform on which benchmarking was done
  - Includes other artifacts from the OS, compiler, and other dependencies
- It is tricky to develop good benchmarks with representative data
  - Estimating best-fit can be tricky at times
- Not effective for computationally challenging problems

Experimental exercise: In this part of the exercise, you will be collecting timing data for the following two different approaches to read information from a text file:

- Using a Scanner: This is a traditional approach to reading data in a sequential manner. That is, to read the nth number, the previous n-1 numbers have to be read.
- Random access: This method *assumes* that each line is of fixed size (an important assumption) and uses a `RandomAccessFile` class to jump to a given index in the file and read the relevant line.

Conduct your experiment in the following manner:

1. Create a Java project in Eclipse.
2. Download the supplied starter code `NumberReader.java` and `fixed_size_nums.txt` to your Java project.
3. Study the starter code while paying attention to the following details:
   a. View the fixed_size_nums.txt in Eclipse and note how each line is of fixed size. This makes it easy for us to find the *n*th line.
   b. Note how the `getNthIntRandom` works using a `RandomAccessFile` and `seek`'s to the desired offset to the read the nth value.
   c. Notice how the `getNthIntRegular` method has to read n-1 values to get to the nth value. This is called <u>serial</u> or <u>sequential</u> access.
   d. Note how the main method uses `System.currentTimeMills` to measure the runtime of the two methods in milliseconds.
4. Based on your code review answer the following questions:
   a. How would you modify the `getNthIntRandom` method if each line is 15-characters wide?

      The named-constant `LINE_LEN` would be modified to `15`.

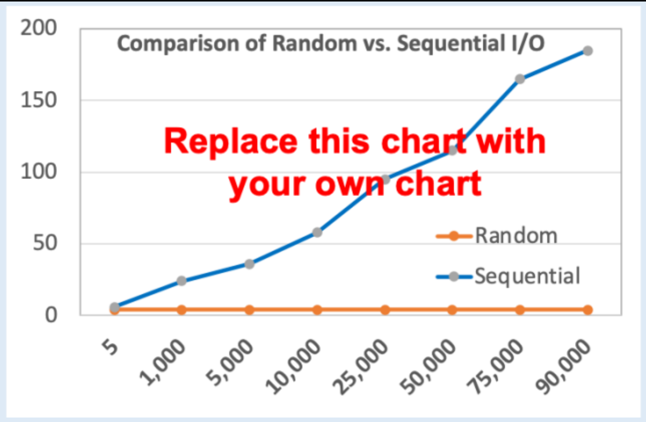
b. What two values does the `main` method use to compute the elapsed time of a method?

The main method subtracts the current time (after method call returns) and starting time (recorded before the method call) to determine the elapsed time of a method.

5. Next run the program and record the time reported for each method in the table below. Use the data to plot a chart.

Observations and chart:

Input *n*	Random read time (ms)	Sequential read time (ms)
5	4	6
1000	4	24
5000	4	36
10000	4	58
25000	4	95
50000	4	115
75000	4	165
90000	4	185



Inference(s):

6. Using the data from the chart estimate the analytical (Big-O) time complexity for the two approaches:

a. Time complexity for random I/O	O(1)
b. Time complexity for sequential I/O	O(n)

7. Many relational databases typically use fixed-size rows (or lines), consisting of fixed size columns. Based on this experimental analysis, is the approach of using fixed-size rows a good design approach? Why or why not?

The experiment showed that using random access to read data from a file provides a constant time access, immaterial of where the data is stored in the file. However, to enable random access we need to know the size of a row/line. Hence, using a fixed-size rows/line in a database is a good design approach.

Part #4: Coding for performance

Estimated time: < 20 minutes

Background: In most programming languages, standard algorithms like binary search and sorting will be built into the standard libraries of the language. You should always prefer to use the built-in implementations. However, in many problem-solving scenarios, you may not have built-in methods and you will need to use first principles to develop your own custom solution. For this, practicing with the underlying concepts of the algorithms will be necessary.

Exercise: Complete this exercise via the following procedure:

1. Create a new Java project in Eclipse
2. Download the supplied starter code SearchSort.java and SearchSortTest.java to your new Eclipse project.
3. Follow the Javadoc comments in the starter code to implement the two methods in SearchSort.java
4. Use the SearchSortTest.java to test your implementation.

Expected output:

```
Testing insert method...
1st insert test passed
2nd insert test passed
3rd insert test passed

Testing countDuplicates method...
1st dupe count test: 2
2nd dupe count test: 7
1st dupe count test: 0

Testing countDuplicates for O(n)...
Done.
```

Part #5: Submit to Canvas via CODE plug-in

Estimated time: < 5 minutes

Exercise: You will be submitting the following files via the Canvas CODE plug-in:

1. This MS-Word document saved as a PDF file – **Only submit PDF file**.
2. The Java source file `SearchSort.java` that you modified in this exercise.

Ensure you actually complete the submission on Canvas by verifying your submission (after you submit)