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

CSE-271: Object-Oriented Programming

Exercise #9

Max Points: 20

Name: Jacob Igel

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?

   When a program attempts to use more space than is available on the call stack (that is, when it attempts to access memory beyond the call stack's bounds, which is essentially a buffer overflow), the stack is said to overflow, typically resulting in a program crash.

   Source: https://en.wikipedia.org/wiki/Stack_overflow#:~:text=When%20a%20program%20attempts%20to,resulting%20in%20a%20program%20crash.

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

```
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. _____Selection Sort_____: 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. _____Quick Sort_____: 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)?

   I would try and use Merge Sort since it is the fastest out of the two.

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)	N characters have to be compared for them to be equal
`int[] list = {1, 2, 3, 4, 5};` `Arrays.binarySearch(list, 5);`	O(log n)	Binary search is O(log n)

`void insert(ArrayList<Integer> list) {`
`  int mid = list.size() / 2;`
`  list.add(mid, -1);`
`}`
	O(n)	Inserting requires copying the amoung of values one spot over for another value to be inserted

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

`int[] list = {1, 2, 3, 4, 5};` `Arrays.sort(list);`	O(n log n)	Quick sorting is used and when know that it uses O(n log n)

`Float get2nd(ArrayList<Float> list) { Collections.sort(list); return list.get(1); }`	O(n log n)	We can tell that this is merge sort which has 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?

   Collections.sort() Operates on **List** Whereas Arrays.sort() Operates on an **Array**.

   Arrays.sort() uses Dual-Pivot Quicksort for Primitive Arrays and MergeSort for sorting array of Objects.

   Source: https://stackoverflow.com/questions/5208133/collections-vs-arrays-regarding-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?

   Well we know that O(n log n) is faster and better than O(n) so we should use Program-β

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)

   Provides an actual performance data and resource consumption

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

   Requires development, which can be time consuming

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 n^{th} 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 n^{th} value.
    c. Notice how the `getNthIntRegular` method has to read n-1 values to get to the n^{th} value. This is called serial or sequential 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?

    LINE_LEN would be modified to 15.

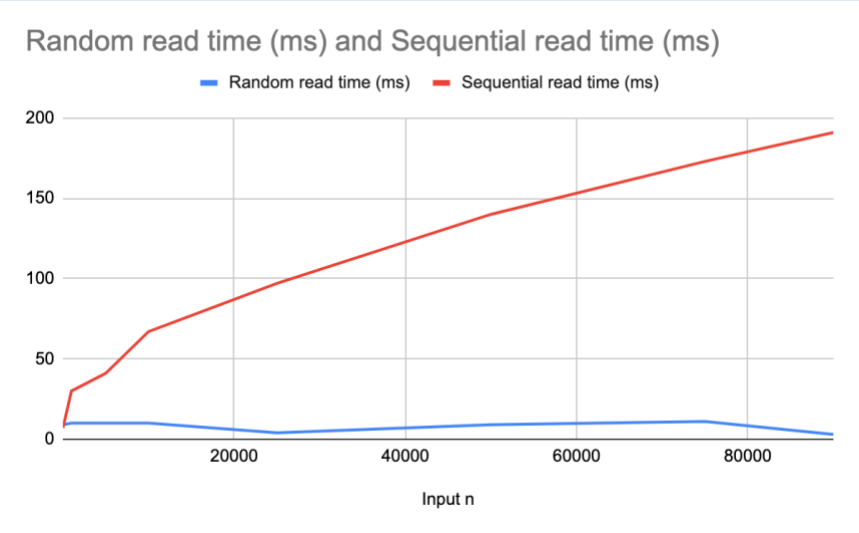
    b. What two values does the `main` method use to compute the elapsed time of a method?

    The main method subtracts the current time and starting time. This determines the total time spend on the 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	9	7
1000	10	30
5000	10	41
10000	10	67
25000	4	97
50000	9	140
75000	11	173
90000	3	191



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?

   Using a fixed-size approach is good because since you know exactly how many rows you have, you are able to use random access. When using random access, you need to know how many rows there are so you can get the time to be consistent.

Part #4: Coding using searching & sorting concepts

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)