Regis University CC&IS
CS310 Data Structures
Programming Assignment 7: Sorting

Problem Scenario

The real estate IT manager was away on a boondoggle conference in this week. He did not leave you anything to do, so you are looking for something to do. All of a sudden, you have a "Eureka" moment, and decided to test different sorting algorithms and see how much time each one takes.

Program Requirements

This will be a **new** Java application, and you will need to create a new project in NetBeans to complete the assignment. Please name the project **CS310<lastname>Sort**.

For example: `CS310SmithSort`

The program must follow all **CS310 Coding Standards** from Content section 1.9.

Java collections will ***not*** be used in this program.

Write a program to compare how long it takes to sort a list, using various sorting methods.

- You will develop methods that will sort the same list with **three** sort methods and compare them.
  Sorting Method 1: Selection Sort or Bubble Sort (you choose one)
  Sorting Method 2: Insertion Sort or Shell Sort (you choose one)
  Sorting Method 3: Merge Sort or Quicksort (you choose one)

- For more accurate results, you will run each sort algorithm three times (each time with a different list), and calculate the average time for the three iterations.
  - Create a 3x3 element 2D **results** arrays that will hold the time it takes to perform each sort.

- Repeat 3 times:

  a. Create a list of numbers to sort.
  Generate 50,000 random numbers between 1 and 100,000 and place them into ***three*** 50,000-element arrays (duplicates **are** allowed). The three arrays should contain identical lists.

  The following pseudo-code describes the basic technique for constructing the test arrays:

```
java.util.Random random = new java.util.Random();

// Create the lists:
Integer[] list1 = new Integer[numElements];
Integer[] list2 = new Integer[numElements];
Integer[] list3 = new Integer[numElements];

// Initialize the values in each list:
for (i = 0; i < numElements; i++)
    list1[i] = list2[i] = list3[i] = random.nextInt(100000);
```

  b. Sort the lists and time how long each sort takes to sort the array using each method.

  Sort the first array using your first sorting method, sort the second array using your second sorting method, and sort the third array using your third sorting method.

Add a means of determining how long each sort took:
Research `System.currentTimeMillis()` or `System.nanoTime()`

Display the timing results as each sort is run. Also place the timing results for each sort into the **results** array.

c. After each sort, verify that the sort produced a list in sorted order.

Write a method that will compare the values in each array cell to the value in the cell next to it, making sure that the higher numbered cell contains a value equal to or higher than the lower numbered cell. If all values are in the correct order, display a message, stating that the sort has been validated. If any list is found to be out of order, display message stating that there was a sorting error, which sort caused the error, and exit the program.

NOTE: This method will help you debug your sorting functions. By the time you turn in your code, the program should not issue any sorting error messages.

Sample Display Output

```
Starting sort #1...
        Bubble Sort time xx.x
        Insertion Sort time xx.x
        Quick Sort time xx.x
           Sorts validated
Starting sort #2...
        Bubble Sort time xx.x
        Insertion Sort time xx.x
        Quick Sort time xx.x
           Sorts validated
Starting sort #3...
        Bubble Sort time xx.x
        Insertion Sort time xx.x
        Quick Sort time xx.x
           Sorts validated
```

- When all of the sorting has been completed without sorting errors, create a report for the averages.

Average the 3 sort times obtained for each sort method that are stored in the results arrays. Create a report, clearly labeled by the sort method.

Sample Report

```
SORTING RESULTS
---------------
                  Run 1     Run 2     Run 3     Average
Bubble Sort        xx.x      xx.x      xx.x        xx.x
Insertion Sort     xx.x      xx.x      xx.x        xx.x
QuickSort          xx.x      xx.x      xx.x        xx.x
```

The report output should be stored in a **`sortResults.txt`** file in an **output** folder created within your project.

Analysis

Based on the Big-O expectations in the online Content and textbook, provide an analysis of your findings. If one sort method should be O(n), while another has a O(n log n), insure your findings reflect these expectations.

Provide a short concise paper, no more than one page, to summarize and analyze the results of your test runs, and state your findings. Be sure to say what unit your timings use (sec, msec, nsec, etc).

Additional Requirements

- Create **Javadoc headers**, and generate **Javadoc files**.
- Your output report will still be written to the **output** folder in your project.
- Add your one page analysis to the documentation folder within the project.
- Add screen shots of **clean compile** of your classes to the documentation folder.

  WARNING: Submittals without the clean compile screenshots will **not** be accepted.
  (This means that programs that do not compile will **not** be accepted)

Program Submission

This programming assignment is due by midnight of the date listed on the **Course Assignments by Week** page.

- Export your project from NetBeans using the same method as you did for previous weeks.
  - Name your export file in the following format:
    `CS310<lastname>Assn<x>.zip`

    For example: **`CS310SmithAssn7.zip`**

- Submit your **.zip** file to the **Prog Assn 7** Submission Folder (located under **Assignments** tab in online course).

  Warning: Only NetBeans export files will be accepted.
  Do not use any other kind of archive or zip utility.

Grading

This program will be graded using the **rubric** that is linked under **Student Resources** page.

WARNING:

*Programs submitted more than 5 days past the due date will **not** be accepted, and will receive a grade of 0.*