Metropolitan State University
ICS 240 - 50: Introduction to Data Structures
Spring 2022

Homework 06: Comparative analysis of Sorting Algorithms
Total points: 40
Out: Tuesday July 26, 2022
Due: 11:59 PM on Tuesday August 9, 2022

Problem Description

The objective of this project is to compare the performance of the quick sort, merge sort and insertion sort algorithms. Basically, you will perform an experiment to run the three sorting algorithms and collect the amount of time it takes for each algorithm to sort lists of varying lengths. Using this data, you will draw a graph to compare and contrast the performance of the various sorting algorithms.

This is a multi-step project. For best results, implement the parts in the order given below.

- **Step 1:** Test the implementation of the three sorting algorithms using integer arrays.
- **Step 2:** create `class Library` to be used to represent the data to be sorted.
- **Step 3:** adapt the implementation of the sorting algorithms to sort library objects.
- **Step 4:** upload data about 17K US libraries from a text file to an array of Library objects.
- **Step 5:** run the sorting algorithms multiple times on arrays with varying sizes and collect the time taken to sort the arrays.
- **Step 6:** write a report to summarize the results of your experiments and summarize your learnings from this project.

Step 1: Sorting Algorithms

1- Create a new project on Eclipse.
2- Create a file called `SortingAlgorithms.java`.
3- Add the implementation of quick sort (slides #131 and #132 from lecture-12-apr-08.pptx).
4- Write a `main` method to test the quick sort algorithm on an array of integers to make sure the algorithm works as expected.
5- Add the implementation of merge sort (slides #98 and #99 from lecture-12-apr-08.pptx). Test the code by sorting an integer array.

6- Add the implementation of insertion sort (slide #68 from lecture-12-apr-08.pptx). Test the code by sorting an integer array.

Step 2: `class Library`

1. Implement a class, called `Library`, as described in UML below.
2. `Library` must implement the `Comparable` interface and the `compareTo()` method must compare the branch names and only the branch names. The comparison must be case insensitive.
3. The `equals()` method must compare the branch names and only the branch names. The comparison must be case insensitive.
4. Be sure to test the `equals()` and `compareTo()` methods before proceeding.

```
Library
- state: String
- branch: String
- city: String
- zip: String
- county: String
- int squareFeet: int
- int hoursOpen: int
- int weeksOpen: int
+ Library(state: String, branch: String, city: String, zip:
String, county: String, squareFeet: int, hoursOpen: int,
weeksOpen: int)
+ getState(): String
+ getBranch(): String
+ getCity(): String
+ getZip(): String
+ getCounty(): String
+ getSquareFeet(): int
+ getHoursOpen(): int
+ getWeeksOpen(): int
+ setState(state: String): void
+ setBranch(branch: String):void
+ setCity(city: String):void
+ setZip(zip: String):void
+ setCounty(count: : String):void
+ setSquareFeet(sqfeet: int): void
+ setHoursOpen(hrsOpen: int): void
+ setWeeksOpen(weeksOpen: int): void
+ equals(Object obj): boolean
+ compareTo(Library): int
+ toString(): String
```

Step 3: Library Sorter

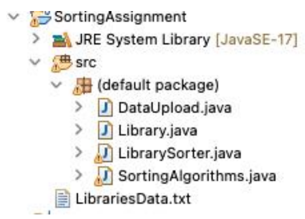
1. Create a new java class `LibrarySorter.java`
2. Copy the quicksort, merge sort and insertion algorithms from the `SortingAlgorithms.java` class to `LibrarySorter.java`
3. Change the sorting algorithms to work with arrays of Library objects instead of array of integers. Remember that libraries are sorted according to compareTo method.
4. Create a `main` method to test the sorting algorithms on short lists of Library objects.

Step 4: Data Upload

1- You are given a data file, called `LibrariesData.txt`, that includes data about 17478 libraries in the US. This data set is collected by the Data is Plural project (https://www.data-is-plural.com/) and more information about this data set can be found at: https://www.imls.gov/research-evaluation/data-collection/public-libraries-survey

2- In this step, you will upload the data from that file to an array of Library objects. You will then sort this array using the various sorting algorithms.

3- Use `import` in Eclipse to move the `LibrariesData.txt` file, into your Eclipse project folder but not into the `src` folder (see screenshot below). This is where the program expects to find the file. If you do not place the file at this location the program will not be able to find it.



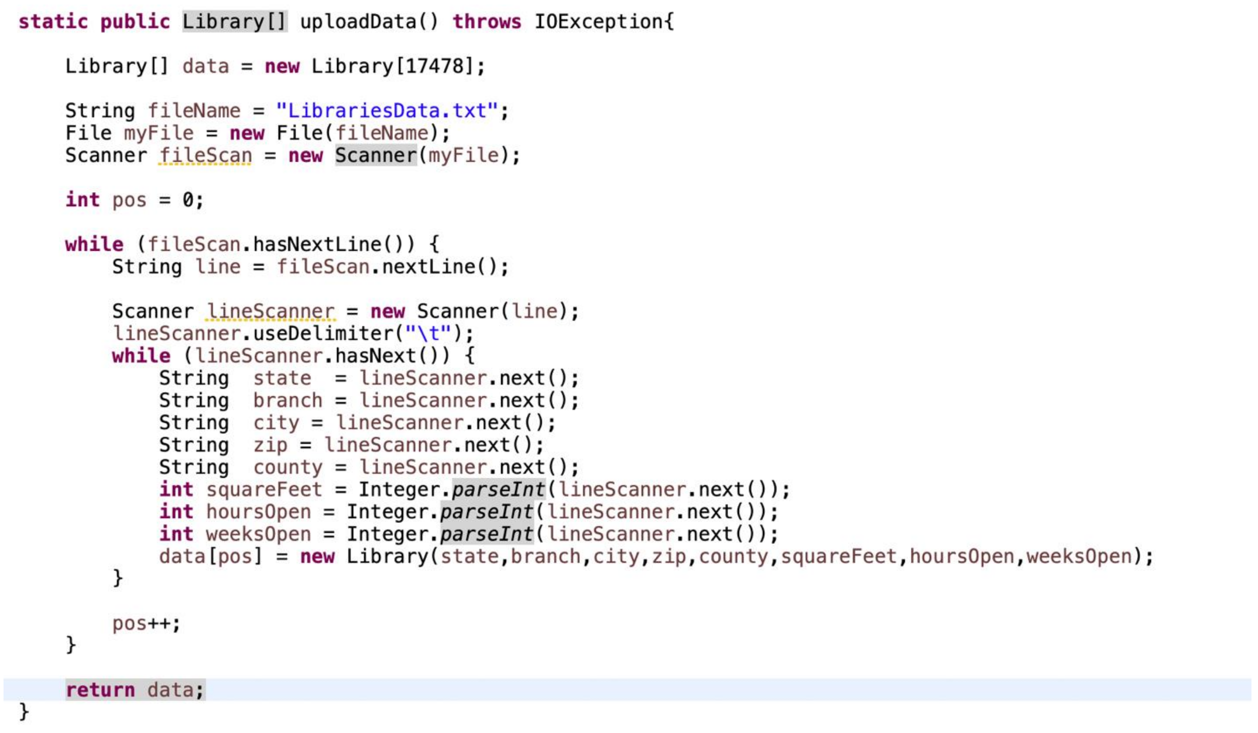
```
SortingAssignment
  JRE System Library [JavaSE-17]
  src
    (default package)
      DataUpload.java
      Library.java
      LibrarySorter.java
      SortingAlgorithms.java
  LibrariesData.txt
```

4- Create a java file, called `DataUpload.java`, with a main method.

5- Write the following method that uploads data from the input file to an array of Library objects and returns that array as output. The method open the data file, read one line at a time, create a Library object from the data read, and add that object to an array. Note that `throws IOException` must be added to the method header the method is using with file IO. Make sure to add the following imorts:

```
import java.io.File;
```

```java
import java.io.IOException;
import java.util.Scanner;
```

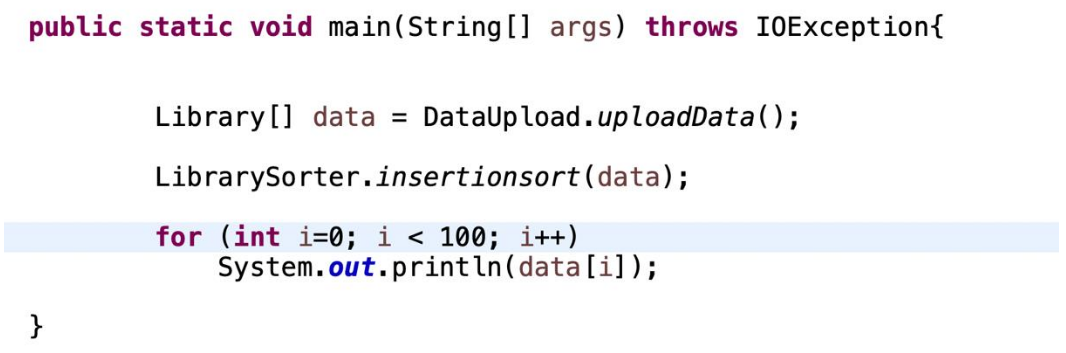


```java
static public Library[] uploadData() throws IOException{

    Library[] data = new Library[17478];

    String fileName = "LibrariesData.txt";
    File myFile = new File(fileName);
    Scanner fileScan = new Scanner(myFile);

    int pos = 0;

    while (fileScan.hasNextLine()) {
        String line = fileScan.nextLine();

        Scanner lineScanner = new Scanner(line);
        lineScanner.useDelimiter("\t");
        while (lineScanner.hasNext()) {
            String  state  = lineScanner.next();
            String  branch = lineScanner.next();
            String  city = lineScanner.next();
            String  zip = lineScanner.next();
            String  county = lineScanner.next();
            int squareFeet = Integer.parseInt(lineScanner.next());
            int hoursOpen = Integer.parseInt(lineScanner.next());
            int weeksOpen = Integer.parseInt(lineScanner.next());
            data[pos] = new Library(state,branch,city,zip,county,squareFeet,hoursOpen,weeksOpen);
        }

        pos++;
    }

    return data;
}
```

6- Call the method from main and print few lines from the returned array to make sure the method works as expected.

Step 5: Sorting Experiment

1. Create a new file, called `SortingExperiment.java`, which is a file that pulls all the pieces together. The main method of this file will first read the Library objects from a file, call the quicksort method to sort an array of Library objects and report the amount of time it takes to do the sort. For example, the following code runs insertion sort algorithm on libraries array.



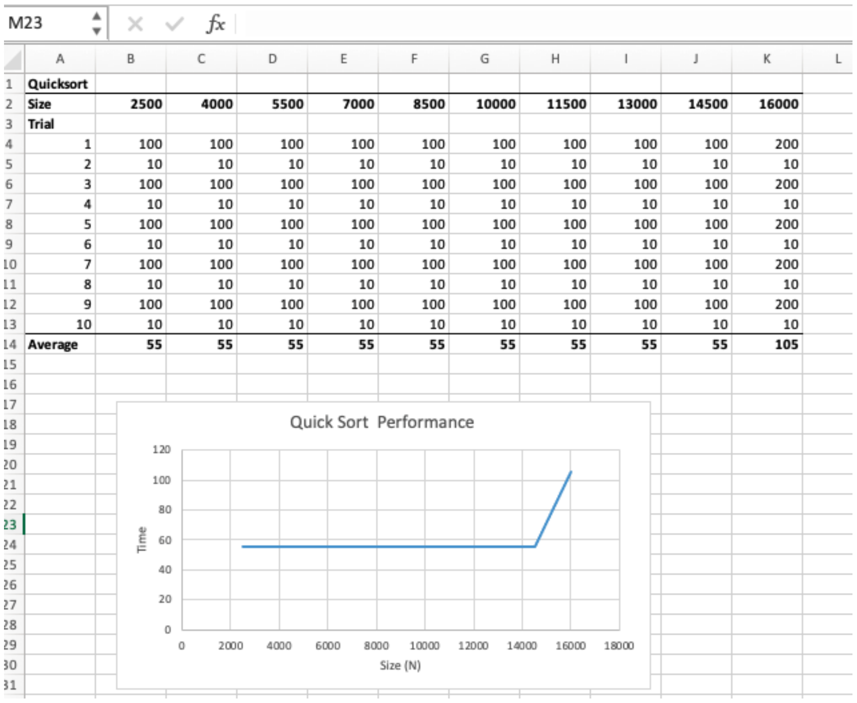
```java
public static void main(String[] args) throws IOException{


        Library[] data = DataUpload.uploadData();

        LibrarySorter.insertionsort(data);

        for (int i=0; i < 100; i++)
            System.out.println(data[i]);

}
```

Quick sort experiment

- Execute quick sort with varying sizes of the array of libraries. Use at least 10 different list sizes ranging from about 1,000 to about 17,000. Depending on how much memory your computer has, you may need to reduce the largest list size.
- For each list size, repeat the experiment for at least 10 times. Record the time takes for each trial and calculate the average of all the trials for each list size. The following code shows how to use the `System.currentTimeMillis` method calculate the time of one trial.

```
public static void main(String[] args) throws IOException{

        Library[] data = DataUpload.uploadData();
        int numToSort = 10000;
        long begin = System.currentTimeMillis();    // start time
        LibrarySorter.quicksort(data,0,numToSort);
        long end = System.currentTimeMillis();      // end time
        System.out.println("Size: " + numToSort + "\tTime in ms: " + (end-begin));

}
```

- Run quicksort for 100 times (10 sizes x 10 trials for each size) and record the numbers in an Excel spreadsheet as follows.



	A	B	C	D	E	F	G	H	I	J	K
1	**Quicksort**										
2	**Size**	**2500**	**4000**	**5500**	**7000**	**8500**	**10000**	**11500**	**13000**	**14500**	**16000**
3	**Trial**										
4	1	100	100	100	100	100	100	100	100	100	200
5	2	10	10	10	10	10	10	10	10	10	10
6	3	100	100	100	100	100	100	100	100	100	200
7	4	10	10	10	10	10	10	10	10	10	10
8	5	100	100	100	100	100	100	100	100	100	200
9	6	10	10	10	10	10	10	10	10	10	10
10	7	100	100	100	100	100	100	100	100	100	200
11	8	10	10	10	10	10	10	10	10	10	10
12	9	100	100	100	100	100	100	100	100	100	200
13	10	10	10	10	10	10	10	10	10	10	10
14	**Average**	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**105**

- IMPORTANT NOTE: you can write for loops to run all the trials at once, HOWEVER, be sure to re-populate the array before every sort. That is, do not sort an array that is already sorted. Read from the file and put fresh Library objects into the array before every sort.

Merge sort experiment

- Execute merge sort with varying sizes of Library array. Use at least 10 different list sizes. Use the same list sizes you used with quicksort.
- Repeat the experiment for each list size at least 10 times.
- This means you will run the merge sort 100 times.
- Record each trial and calculate the average of all the trials for each list size.
- Add another sheet to the excel file to include rows, columns and labels for the merge sort data. Enter your data in this spreadsheet.
- Add a line to the graph to represent the results of the merge sort experiment. To be clear, you will have ONE graph with TWO lines and NOT TWO graphs.

Insertion sort experiment

- Execute insertion sort with Library lists of varying sizes. Use at least 10 different list sizes.
- Use the same list sizes you used with quicksort and merge sort.
- Repeat the experiment for each list size at least 10 times.
- This means you will run the merge sort 100 times.
- Record each trial and calculate the average of all the trials for each list size.
- Add another sheet to your excel file to include rows, columns and labels for the insertion sort data. Enter your data in this spreadsheet.
- Add a line to the graph to represent the results of the insertion sort experiment. You will now have ONE graph with THREE lines.
- Make sure each line in the graph is a different color and/or texture.
- Add a legend to the chart so the viewer can tell what each line means.

Step 6: Report

Write a brief report on the experiment. Address the following points:

1. What is the expected (theoretical) performance of each sort? (For example, O(N), O(N^2), O(lg N), O(N lg N).)
2. What is the actual performance of each sort as revealed by your data? Hint: Look at the curves in your graph.
3. According to your data, which sort is best? Which is worst?
4. What did you learn from doing this project?

What to submit?

Upload to D2L ONLY ONE zip file that includes the following files:

1- SortingAlgorithms.java
2- Library.java
3- LibrarySorter.java
4- DataUpload.java
5- SortingExperiment.java
6- An excel sheet that includes the numbers reported from you're the experiemnts as explained above.
7- A word file that includes answers to the questions in Step 6: Report.

Grading

Your grade in this assignment is based on the following:

- Your zip file is submitted on time.
- Your submission meets all specifications as described above.
- The program is robust with no runtime errors or problems.
- **Raw data and averages are recorded in a spreadsheet. A graph of the results is provided in the spreadsheet.**
- **A brief report that answers the questions about the experiments.**
- You must use the exact same name (including upper case and lower case letter) for all methods as specified in the description above.
- You must follow the method requirements in terms of the number, order and data type of input parameters and the output data type as given.