جامعة خليفة
Khalifa University

Module Name:	Data Structures and Algorithms
Module Code:	COSC 310 / ECCE 342
Term:	Spring 2024
Instructor:	Dr. Majid Khonji
Teaching Assistant:	Mahmoud Said Elmezain
TA Email:	100059645@ku.ac.ae
Issue Date:	Mar 25, 2024
Submission Deadline:	Wed Apr 24, 2024 @ 11:59 PM
Assignment Type:	Individual

AIM

The aim of this assignment is to build a class (or more) and a driver class to test and analyze the implementation and performance of various sorting algorithms.

TASKS

TASK 1. READING DATA. [10 MARKS]

Create a Java class that can retrieve information on 1,000 cars from a text file named `"cars.csv"`. This file uses the Comma Separated Values (CSV) format, meaning each row of data is separated by commas. The file includes a header row specifying the names of the three columns: VIN, Name, and DOB. Your task is to design and implement the class that can read and extract data from this file.

- **VIN (Vehicle Identification Number).** Identifier of length 17 characters (letters and numbers), unique for each car.
- **Name.** The first name and last name of the car owner. The first and last names have no spaces, except the space separating them.
- **DOB (Date of Birth).** The day the car owner was born in. Ranges from `1980-01-03` to `2009-12-18`, inclusive. Formatted as `YYYY-MM-DD`.

cars.csv

```
VIN,Name,DOB
48SGQWKGY9KH3AVHQ,Henrieta Byrann,1986-01-12
A94479OH3LM258DIQ,Ulrica Doug,1987-02-16
...
FW0YOYP13SOP0S8EH,Rozalin Ezra,1980-08-11
```

Your program must read and store the data of the cars in an array of objects. Furthermore, the method should write the cars data into a numbered and neatly formatted text file called `formatted_cars.txt`, as shown in the sample below. For good formatting, make sure to use formatted printing `printf()`. Notice the change in order of columns.

The data in "VIN" and "No." columns should be modified according to the last two digits of your KU ID. For example, if your KU ID is 1000123456, then the ID would be 56.

The "No." column should start from "ID+1" and end at "ID+1000", for the aforementioned example, starting at 57 and ending at 1057. And in the VIN column, the last two digits of the VIN number should be "ID". For instance, if the VIN number is "O0XYZRZSUYNYHQ3EK" and ID is "56", then the VIN number should be "O0XYZRZSUYNYHQ3EK56".

formatted_cars.txt

```
No.       Name              DOB               VIN
1+ID      Abigail Orville   1983-04-25        O0XYZRZSUYNYHQ3EKID
2+ID      Adaline Laughton  1982-08-16        NDUI28G67EJMBN6D2ID
...
1000+ID   Zuzana Kylie      1983-01-21        KOSH38POVQED7QN09ID
```

TASK 2. SORTING DATA. [20 MARKS]

Create a class that can sort a set of car data in ascending order based on the name of their owner. This sorting class must correctly implement the following five sorting algorithms:

- `BubbleSort()` (Short bubble sort algorithm)
- `InsertionSort()`
- `QuickSort()` (Optimized such that it has no unnecessary recursive calls, e.g. when `F >= L`)
- `HeapSort()`
- `MergeSort()`

Apply the sorting algorithms to the array of car objects, and save the sorted arrays into the following files, numbered, and formatted neatly:

- `bsorted.txt` (for bubble sort)
- `isorted.txt` (for insertion sort)
- `qsorted.txt` (for quick sort)
- `hsorted.txt` (for heap sort)
- `msorted.txt` (for merge sort)

Make sure that you maintain the integrity of the data after sorting.

TASK 3. QUANTITATIVE ANALYSIS. [30 MARKS]

Adjust the sorting algorithms to count the number of operations (type of operations detailed below in the tables) performed. And change the calling function such that you record the time the algorithms take.

You will need to create 5 arrays of cars (number of elements detailed below in the tables) and apply the sorting algorithms on these unsorted arrays. Then again, you need to re-apply the sorting algorithms on the sorted arrays.

TABLE 1. BUBBLE SORT ON UNSORTED ARRAY.

# Elements	# Comparisons	# Swaps	Time (milliseconds)
250	**31070**	**15804**	**31070**
500			
750			
1,000			

TABLE 2. BUBBLE SORT ON SORTED ARRAY.

# Elements	# Comparisons	# Swaps	Time (milliseconds)
250			
500			
750			
1,000			

TABLE 3. INSERTION SORT ON UNSORTED ARRAY.

# Elements	# Comparisons	# Assignments	Time (milliseconds)
250			
500			
750			
1,000			

TABLE 4. INSERTION SORT ON SORTED ARRAY.

# Elements	# Comparisons	# Assignments	Time (milliseconds)
250			
500			
750			
1,000			

TABLE 5. QUICK SORT ON UNSORTED ARRAY.

# Elements	# Comparisons	# Swaps	Time (milliseconds)
250			
500			
750			
1,000			

TABLE 6. QUICK SORT ON SORTED ARRAY.

# Elements	# Comparisons	# Swaps	Time (milliseconds)
250			
500			
750			
1,000			

TABLE 7. HEAP SORT ON UNSORTED ARRAY.

# Elements	# Comparisons	# Swaps	# Reheap downs	Time (milliseconds)
250				
500				
750				
1,000				

TABLE 8. HEAP SORT ON SORTED ARRAY.

# Elements	# Comparisons	# Swaps	# Reheap downs	Time (milliseconds)
250				
500				

750				
1,000				

TABLE 9. MERGE SORT ON UNSORTED ARRAY.

# Elements	# Comparisons	# Merge calls	Time (milliseconds)
250			
500			
750			
1,000			

TABLE 10. MERGE SORT ON SORTED ARRAY.

# Elements	# Comparisons	# Merge calls	Time (milliseconds)
250			
500			
750			
1,000			

TASK 4. QUALITATIVE ANALYSIS. [30 MARKS]

For each of the sorting algorithms, report your results from the previous task as to why and how the algorithm performed as the number of elements increased and as the data got sorted.

You need to justify the retrieved number of operations in the unsorted case of 1,000 elements and estimate how these numbers can be calculated. Moreover, correlate the time the algorithms took with its big-O time complexity.

Finally, investigate whether heap sort is a stable algorithm or not, and describe what that means.

TASK 5. DISCUSSION. [10 MARKS]

Compare the performance of the four sorting algorithms in terms of their execution time and big-O time complexity – not $T(n)$ – for both sorted and unsorted arrays. Discuss and justify how each algorithm can be optimal in a certain use case.

NOTES

HINTS

- Make sure that when you sort the cars based on a certain column value, you also sort the other parts of the car data.
- Note that you should rewrite the sorting algorithms given to you in class to cater for ordering based on the age, name, and VIN.
- You need to write a driver class to use the algorithms for testing.
- You need to write the program in an object-oriented style (must have at least two classes).

CODE

- All submitted code should be properly documented and written in a good programming style with comments and proper indentation.
- You are only allowed to use code from the notes in class or lab with no change of structure or naming style (no internet code will be accepted).

SUBMISSION

- Submit screenshots of the results in the report (output files and console results).
- Your report must be a single Word document, containing all code, screenshots of the first and last 10 cars for each run. Name your report document as `FIRSTNAME_ID.docx`.
- Compress the `.java` files into a `FIRSTNAME_ID.zip` file and submit the code along the Word document.
- Delayed Submissions will be penalized by 5% points a day with a maximum delay of 5 days. Submissions after that will not be accepted.

WARNINGS

- No collaboration between students is allowed and any detection of plagiarism from each other or from the internet will be penalized with zero mark for all of those involved and will be reported to the Dean's office to take further disciplinary actions.