CCDSALG

Project 1 – Comparing Sorting Algorithms

Major Details

Groupings:	At most 3 members in a group
Deadline:	February 15, 2023 (W) 10:30 AM
Percentage:	15%
Submission guidelines:	Submit the `zip` file to AnimoSpace
Filename format:	`CCDSALG-Project1-<Section>-Group<#>.zip`

Deliverables

`Zip` file containing:

- Program source codes – `c` files
- Documentation – `PDF` file

Specifications

Task 1 – Information Gathering

- Search and gather existing implementations (i.e., source codes) of different sorting algorithms that are made available for free on the internet. You need to gather <u>at least six sorting algorithms</u>. You must include the following algorithms in your experiments: (1) bubble sort, (2) insertion sort, (3) selection sort, and (4) merge sort. It is up to you to choose the other algorithms.
- Make sure that the sorting algorithms are all written in C programming language.
- Cite the site where you got the codes, and if available, the author/creator of the source code.
- Study the implementation and test and verify that it runs correctly.

Task 2 – Coding

- Place the code for each sorting algorithm in the corresponding header file provided in the project files. See the table below:

Filename	Function Prototype	Sorting Algorithm
`bubbleSort.h`	`void bubbleSort(int A[], int n, double *dCounter)`	Bubble sort
`insertionSort.h`	`void insertionSort(int A[], int n, double *dCounter)`	Insertion sort
`selectionSort.h`	`void selectionSort(int A[], int n, double *dCounter)`	Selection sort
`mergeSort.h`	`void mergeSort(int A[], int n, double *dCounter)`	Merge sort
`sort5.h`	`void sort5(int A[], int n, double *dCounter)`	<Sorting algorithm 5>
`sort6.h`	`void sort6(int A[], int n, double *dCounter)`	<Sorting algorithm 6>

Table 1. Mapping of the different sorting algorithms to its corresponding filename and function prototype.

- You are <u>not allowed</u> to modify the function prototypes provided in the header files for sorting algorithms – no additional parameter may be declared, nor existing parameters may be removed. All sorting algorithms accept arguments to the same set of parameters listed below:

Parameter	Definition
`int A[]`	Array to be sorted
`int n`	Size of the array to be sorted
`double *dCounter`	Pointer to a counter variable for critical parts of the code

Table 2. Parameters of the function prototypes for all sorting algorithms. No modification – adding or removing – is allowed.

- You can declare additional functions to aid in the implementation of each sorting algorithm. Additional functions may be declared inside the header file of a sorting algorithm (i.e., `bubbleSort.h`, `insertionSort.h`, etc.) or in `main.h`.
- Modify the source code such that each sorting algorithm will have a counter variable to count the number of times it goes through the critical parts of the code. These are the parts of the code that are affected as we increase the number of items *N* in the list of randomly generated numbers.
- Implement the function `generateData(A, n)` in the provided header file `generateData.h` where `A` corresponds to the array and `n` is the array size. It should automatically generate the random *N* values to be sorted.
- Implement the `main()` function in the `c` file `main.c`. The `main()` function should do the following steps:

```
For each value of N(= 1024, 2048, ... ):
        Call generateData(A, n)

        For M = 1 to at least 10 (number of runs):
                For each sorting algorithm:
                        (1) Get the start CPU time using getTime() function in
                        timer.h.
                        (2) Call the sorting function.
                        (3) Get the end CPU time using getTime() function
                        in timer.h.
                        (4) Compute the machine execution time (MET) using the
                        getElapsed() function in timer.h. This returns the
                        elapsed time in seconds.
                        (5) Record the MET.
                        (6) Record the counter value.

        Compute and display average MET (in milliseconds) per algorithm.
        Compute and display average counter value per algorithm.
```

Figure 1. The `main()` function algorithm

- The `main()` function should call the functions for the sorting algorithms by providing arguments to the parameters of the function prototypes detailed in Table 1. Thus, you are required to use the function prototypes in Table 1.

Task 3 – Testing

- Run each sorting algorithm for different and increasing values of N as a power of 2 (for example: $N = 1024, 2048, 4096, \dots, 65536, 131072, \dots$).
- For each value of N, execute each sorting algorithm at least $M = 10$ times. Please refer to the description of the `main()` function in Figure 1.
- For each value of N, record the corresponding average MET and the average counter value based on M runs of each sorting algorithm.
- Try to increase N to a value that can still be handled by all sorting algorithms.
- For each N, all sorting algorithms must use same input array data values M times. Please refer to the description of the `main()` function in Figure 1.
- For fair comparison, all tests should be made using the same machine.
- The program should run (and be tested) using the TDM-GCC compiler (available in Dev C++). C99 mode should be disabled.

Task 4 – Documentation

- Document your experiment results following the template in the file `Documentation.docx`.
- Summarize the results into a table.

- Visualize the results for the average MET as line graphs, where the x-axis is for N and the y-axis is for the average MET of each sorting algorithm. The average MET of all sorting algorithms should be combined in a single line graph.
- Visualize the results for the average counter value as line graphs, where the x-axis is for N and the y-axis is for the average counter value of each sorting algorithm. The average counter variable of all sorting algorithms should be combined in a single line graph.
- Discuss your observations based on the experiments. Explain the comparison between the average MET and the rate of growth, in terms of the average counter value, of each algorithm. Use the comparison table and the graphs to support your discussion.

Required Program Interaction

There should be no program interaction. Upon running the code, the program should display the results in the console. No input will be asked from the user.

See sample run below, where $N = 1024, 2048$. Parts highlighted in yellow are the average MET (in milliseconds) and the average counter value for each sorting algorithm (here displayed with dummy values), which are computed by the `main()` function automatically. Do not forget to convert the output of the `getElapsed()` function from seconds to milliseconds. Your `main()` function should exactly resemble this output, with the values highlighted in yellow replaced with their actual values computed by the `main()` function. Additional displayed statements will result in deductions.

```
-- N: 1024 --

Bubble Sort:
Average MET: x.xxxxxx milliseconds
Average counter value: x

Insertion Sort:
Average MET: x.xxxxxx milliseconds
Average counter value: x

Selection Sort:
Average MET: x.xxxxxx milliseconds
Average counter value: x

Merge Sort:
Average MET: x.xxxxxx milliseconds
Average counter value: x

Sort 5:
Average MET: x.xxxxxx milliseconds
Average counter value: x

Sort 6:
Average MET: x.xxxxxx milliseconds
```

```
Average counter value: x

-- N: 2048 --

Bubble Sort:
Average MET: x.xxxxxx milliseconds
Average counter value: x

Insertion Sort:
Average MET: x.xxxxxx milliseconds
Average counter value: x

Selection Sort:
Average MET: x.xxxxxx milliseconds
Average counter value: x

Merge Sort:
Average MET: x.xxxxxx milliseconds
Average counter value: x

Sort 5:
Average MET: x.xxxxxx milliseconds
Average counter value: x

Sort 6:
Average MET: x.xxxxxx milliseconds
Average counter value: x
```

Figure 2. Running the `main()` function with $N = 1024,\ 2048$.

Working With Groupmates

For this project, you are encouraged to work in groups of at most 3 members. Make sure that each member of the group has approximately the same amount of contribution for the project. Problems with groupmates must be discussed internally within the group, and if needed, with the lecturer.

Deliverables

Submit a `zip` file containing the source code files and a `PDF` file of the documentation via AnimoSpace. Do not include any executable file in your `zip` file submission.

Academic Honesty Policy

Honesty policy applies. You should explicitly acknowledge the source, i.e., the author (if available) and the URL of the website from where you got the implementation of a sorting algorithm. Include this acknowledgement as a comment in the first few lines of the source codes and in the documentation (see `Documentation.docx`).

The student handbook states that (Sec. 5.2.4.2):

"Faculty members have the right to demand the presentation of a student's ID, to give a grade of 0.0, and to deny admission to class of any student caught cheating under Sec. 5.3.1.1 to Sec. 5.3.1.1.6. The student should immediately be informed of his/her grade and barred from further attending his/her classes."

The student handbook also states that (Sec. 10.3):

A student caught cheating, as defined in Sec. 5.3.1.1., shall be penalized with a grade of 0.0 in the requirement or in the course, at the discretion of the faculty member, without prejudice to an administrative sanction. In cases of alleged cheating, the faculty member should report the incident to the Student Discipline Formation Office (SDFO).

<u>RUBRIC FOR GRADING</u>

<table>
<tr><th>Criteria</th><th colspan="6">Ratings</th><th>Points</th></tr>
<tr><td>Task 1 – Gathering</td><td colspan="2">COMPLETE
5 pts

Gathered 6 sorting algorithms, including bubble sort, insertion sort, selection sort, and merge sort.</td><td colspan="2">INCOMPLETE
2 pts

Gathered at least the 4 required sorting algorithms.</td><td colspan="2">NO MARKS
0 pt

Gathered less than the 4 required sorting algorithms.</td><td>5 pts</td></tr>
<tr><td>Task 1 – Working Algorithms</td><td colspan="3">COMPLETE
5 pts

<u>All</u> 6 Sorting algorithms are working correctly.</td><td colspan="3">NO MARKS
0 pt

Some sorting algorithms are not working correctly.</td><td>5 pts</td></tr>
<tr><td>Task 1 – Proper Citations</td><td colspan="3">COMPLETE
2 pts

Sources/Authors of all 6 sorting algorithms are properly cited.</td><td colspan="3">NO MARKS
0 pt

Sources/Authors of some sorting algorithms are not cited.</td><td>2 pts</td></tr>
<tr><td>Task 2 – Generate Data</td><td colspan="3">COMPLETE
5 pts

Correct implementation and use of the <code>generateData(A, n)</code> function.</td><td colspan="3">NO MARKS
0 pt

Did not implement nor use the <code>generateData(A, n)</code> function. Executed the experiments using hardcoded values.</td><td>5 pts</td></tr>
<tr><td>Task 2 – main() function</td><td colspan="2">COMPLETE
10 pts

Correct implementation of the <code>main()</code> function. All sorting</td><td colspan="2">INCOMPLETE
5 pts

The <code>main()</code> function asks for the value of N from the user.</td><td colspan="2">NO MARKS
0 pt
Implemented the <code>main()</code> function inside the code of each sorting algorithm.</td><td>10 pts</td></tr>
</table>

	algorithms are executed using increasing values of N. All algorithms are sorting the same set of randomly generated numbers for each run for each value of N.	Sorting algorithms are sorting different set of randomly generated number for each run.	Some sorting algorithms are sorting an already sorted array.	
Task 2 – Frequency Count	**COMPLETE** **10 pts** Correctly modified the code of <u>all</u> 6 sorting algorithms to get the frequency count of critical parts of the code.	**INCOMPLETE** **5 pts** Correctly modified the code of <u>some</u> sorting algorithms to get the frequency count of critical parts of the code.	**NO MARKS** **0 pt** Did not modify any sorting algorithms to get the frequency count of critical parts of the code.	10 pts
Task 3 – MET	**COMPLETE** **15 pts** Correctly recorded and displayed the average MET for <u>all</u> 6 sorting algorithms and for varying sizes of N.	**INCOMPLETE** **7 pts** Correctly recorded and displayed the average MET for <u>some</u> sorting algorithms and for varying sizes of N.	**NO MARKS** **0 pt** Did not record nor display the average MET for any sorting algorithm and for varying sizes of N.	15 pts
Task 3 – Counter Value	**COMPLETE** **15 pts** Correctly recorded and displayed the average counter value for <u>all</u> 6 sorting algorithms and for varying sizes of N.	**INCOMPLETE** **7 pts** Correctly recorded and displayed the average counter value for <u>some</u> sorting algorithms and for varying size of N.	**NO MARKS** **0 pt** Did not record nor display the average counter value for any sorting algorithm and for varying sizes of N.	15 pts

Task 3 – M	**COMPLETE** **2 pts** All 6 sorting algorithms are executed at least $M = 10$ times for each N and is properly indicated in the document.		**NO MARKS** **0 pt** Some sorting algorithms are not executed at least $M = 10$ times for each N. M is not properly indicated in the document.	2 pts
Task 4 – Line Graphs for MET	**COMPLETE** **8 pts** Results for average MET of all 6 sorting algorithms are properly and correctly represented in one line graph.	**INCOMPLETE** **3 pts** Results for average MET of some sorting algorithms are properly and correctly represented in one line graph. Results for average MET of sorting algorithms are represented in different line graphs.	**NO MARKS** **0 pt** Line graphs for average MET of sorting algorithms are not present in the document.	8 pts
Task 4 – Line Graphs for Counter Value	**COMPLETE** **8 pts** Results for average counter value of all 6 sorting algorithms are properly and correctly represented in one line graph.	**INCOMPLETE** **3 pts** Results for average counter value of some sorting algorithms are properly and correctly represented in one line graph. Results for average counter value of sorting algorithms are represented in different line graphs.	**NO MARKS** **0 pt** Line graphs for average counter value of sorting algorithms are not present in the document.	8 pts

Task 4 – Discussions	COMPLETE 15 pts Discussions are well written and well-founded. Discussions explain the comparison between the average MET and the rate of growth, in terms of the average counter value, of <u>all</u> 6 sorting algorithms.	INCOMPLETE 7 pts Discussions could be improved. Discussions explain the comparison between the average MET and the rate of growth, in terms of the average counter value, of <u>some</u> sorting algorithms.	NO MARKS 0 pt No discussion was presented in the document. Discussions are based on <u>incorrect experiments</u>.	15 pts
			Total points:	100