CCDSALG Term 3, AY 2019 – 2020

Project 1 – Comparing Sorting Algorithms

Groupings: At most 3 members in a group
Deadline: August 27, 2020 (Thursday) 11:59 PM
Percentage: 10% (part of the 45% for Research Work/Machine Project)

Deliverables:
Zip file containing:

- Program source codes – c files
- Documentation – PDF file (name your documentation file as GroupName.PDF)

Submission guidelines: Submit the `zip` file to AnimoSpace
Filename format: GroupName.zip

Documentation Formatting Guidelines:

Font Size:	11
Line Spacing:	Single
Margin:	1″
Paper Size:	Letter

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 **at least six sorting algorithms**. 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 the same 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

- Modify the source codes 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.

- The source code for each sorting algorithm must be saved on a separate file. For example, the code for merge sort function will be in the file "`merge.c`". This file should not contain a main() function.
- Create a separate source code file that will contain a single function `GenerateData(A, N)` where `A` corresponds to the array and `N` is the array size. It should automatically generate the random N values to be sorted.
- Create a separate source file that will contain the `main()` function. This source file should `#include` the other files (i.e., for generating data, and sort functions). The `main()` function should do the following:
    - 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:
                - Get the start CPU time
                - Call the sort function; for example, `InsertionSort(A, N);`
                - Get the end CPU time
                - Compute the machine execution time (MET) = end CPU time – start CPU time
                - Record the MET
                - Record the counter value
            - Compute and record the average MET
            - Compute and record the average counter value
- Please refer to the accompanying file "`get-cpu-time.c`" on how to determine the CPU time.

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, …, 65536, 131072, …).
- For each value of N, execute each sorting algorithm at least $M = 10$ times (or more).
- For each sorting algorithm, record the corresponding average MET and the average counter value based on M runs for each value of N.
- Try to increase N to a value that can still be handled by the implementation.
- **For each N, all sorting algorithms must use same input array data values – refer to Task 2 description.**
- **For fair comparison, all tests should be made using the same machine.**

Task 4 – Documentation

- Document your experiment results following the template in the accompanying file "`Documentation Template.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.

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. As specified in Task 2, each sorting algorithm should have its own source code file. **DO NOT INCLUDE ANY EXECUTABLE FILE in your zip file submission.**

NON-SUBMISSION POLICY

Non-submission of the project by the due date will result to your deferral for the course. Based on the existing university policy, you will be given the next trimester to complete a missing requirement.

HONESTY POLICY AND INTELLECTUAL PROPERTY RIGHTS

Honest 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 documentations (see "`Documentation Template.docx`").

According to the handbook (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."

In case you implemented a sorting algorithm, state this explicitly in the source codes and in the documentation.

RUBRIC FOR GRADING

Criteria	Ratings			Points
Task 1	**COMPLETE** **5 pts** Gathered 6 sorting algorithms, including bubble sort, insertion sort, selection sort, and merge sort.	**INCOMPLETE** **2 pts** Gathered at least the 4 required sorting algorithms.	**NO MARKS** **0 pt** Gathered less than the 4 required sorting algorithms.	5 pts
	COMPLETE **5 pts** All 6 Sorting algorithms are working correctly and are written using the same programming language	**INCOMPLETE** **2 pts** Sorting algorithms are written using different programming languages.	**NO MARKS** **0 pt** Some sorting algorithms are not working correctly.	5 pts
	COMPLETE **2 pts** Sources/Authors of all 6 sorting algorithms are properly cited.		**NO MARKS** **0 pt** Sources/Authors of some sorting algorithms are not cited.	2 pts
Task 2	**COMPLETE** **5 pts** Correct implementation and use of the `GenerateData(A, N)` function.		**NO MARKS** **0 pt** Did not implement nor use the `GenerateData(A, N)` function. Executed the experiments using hardcoded values.	5 pts
	COMPLETE **10 pts** Correct implementation of the `main()` function. All sorting algorithms are to be executed using increasing values of N.	**INCOMPLETE** **5 pts** Sorting algorithms are sorting different set of randomly generated number for each run.	**NO MARKS** **0 pt** Implemented the `main()` function inside the code of each sorting algorithm.	10 pts

	All algorithms are sorting the same set of randomly generated numbers for each run.			
	COMPLETE **10 pts** Correctly modified the code of all 6 sorting algorithms to get the frequency count of critical parts of the code.	**INCOMPLETE** **5 pts** Correctly modified the code of some 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	**COMPLETE** **15 pts** Correctly recorded the average MET for all 6 sorting algorithms and for varying sizes of N.	**INCOMPLETE** **7 pts** Correctly recorded the average MET for some sorting algorithms and for some varying size of N.	**NO MARKS** **0 pt** Did not record the average MET for any sorting algorithm and for varying sizes of N.	15 pts
	COMPLETE **15 pts** Correctly recorded the average counter value for all 6 sorting algorithms and for varying sizes of N.	**INCOMPLETE** **7 pts** Correctly recorded the average counter value for some sorting algorithms and for some varying size of N.	**NO MARKS** **0 pt** Did not record the average counter value for any sorting algorithm.	15 pts
	COMPLETE **3 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.		3 pts
Task 4	**COMPLETE** **10 pts** Results for average MET of all 6 sorting	**INCOMPLETE** **5 pts** Results for average MET of some sorting	**NO MARKS** **0 pt** Line graphs for average MET of sorting	10 pts

	algorithms are properly represented as line graphs.	algorithms are properly represented as line graphs.	algorithms are not present in the document.	
	COMPLETE **10 pts** Results for average counter value of all 6 sorting algorithms are properly represented as line graphs.	**INCOMPLETE** **5 pts** Results for average counter value of some sorting algorithms are properly represented as line graphs.	**NO MARKS** **0 pt** Line graphs for average counter value of sorting algorithms are not present in the document.	10 pts
	COMPLETE **10 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 all 6 sorting algorithms.	**INCOMPLETE** **5 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 some sorting algorithms.	**NO MARKS** **0 pt** No discussion was presented in the document.	10 pts
			Total points:	100