CSCI 3330 Project 2

Analyzing and verifying asymptotic time complexity of sorting algorithms

---

Introduction: In data science, people extract information from collected data. In this project, you are asked to design and develop a test suite that collects data to verify asymptotic time complexity of four selected sorting algorithms including bubble sort, quick sort, merge sort, and another one of your choice.

Project objectives: Through completing this project, you should be able to

- Specify the requirements of a testing suite that examines asymptotic time complexity in the best, worst, and average cases for a set of selected sorting algorithms.
- Design the test suite to meet the specified requirements with appropriate I/O management including input data generation and output data collection.
- Implement your design as a software solution that obtains experimental data for analyzing computational efficiency of selected sorting algorithms.
- Analyze collected experimental data and verify asymptotic time complexity of each selected sorting algorithms in its best, worst, and average cases.
- Enhance oral and writing communication skills through effective communication within a working group to achieve a common goal.

Project requirements:

1. Four different sorting algorithms need to be examined including bubble sort, merge sort, quick sort, and another one of your choice. An online reference of some sorting algorithms is available at https://medium.com/@george.seif94/a-tour-of-the-top-5-sorting-algorithms-with-python-code-43ea9aa02889.
2. For each of the selected algorithms, the test suite should examine its computational performance in the best, worst, and average cases.
3. Performance data should be collected and analyzed to verify asymptotic time complexity for each of the selected algorithms in its best, worst, and average cases.

Solution design:

1. To examine the performance of each of the selected sorting algorithms, you need to implement the algorithm correctly.
2. To examine the asymptotic time complexity of the selected sorting algorithm experimentally, you should increase n, the size of input data, incrementally in equal magnitude, say 100, 1000, 10,000, 100,000, etc. Collecting the run time for each given n, you obtain a sequence of T(n). From which, you may verify asymptotic time complexity of the sorting algorithm.
3. The best, worst and average cases for a selected sorting algorithm depends not only on the size but also on the property of input data. For instance, the bubble sort algorithm uses significantly different time to sort an ordered array and a reversely ordered array of exactly the same size. You should generate input data not only in various sizes but also to match the best, worst, and average cases.

4. You need to collect performance data for further analysis. Some of the algorithms are quadratic. It may take long time to sort a large size array. Therefore, you need to write the data in a file in an organized manner for further analysis instead of interactive I/O.
5. Your project involves multiple sorting algorithms in different cases with input data in various sizes. You need to integrate solutions of subtasks into a single test suite, i.e. a main program.

Software implementation and testing:

1. Implement each functional unit in your design and verify if it meets the expected requirements.

   Hint: You may want to pass an input dataset as an argument of a function, and write the performance data to a file.
2. Test each functional unit to see if it meets the requirements. Your computer may encounter system errors when sorting a large size array recursively. In such a case, you may adjust the system default parameter or terminate the execution with explanation in your report.
3. You need to integrate all functional units as a suite named main. Test it to see if it generates all expected experimental data for further analysis.

Analyzing computational results with documentation:

1. Beyond properly document your software, you need to document your work as a project report.
2. In the report, you need to analyze collected data and verify that your experimental data support the asymptotic time complexity in theory.

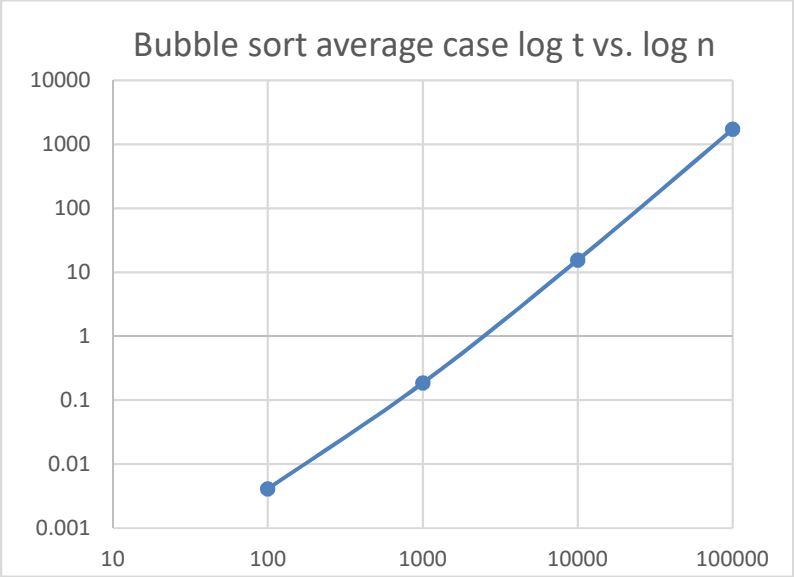
   For example, here is a sample for analyzing bubble sort in its average case:

   To examine the time complexity of bubble sort algorithm in its average case, we use *randomly* generated integer arrays. The table below lists the runtime for each given size array.

n	T(n)
100	0.0041
1,000	0.18518
10,000	15.4804
100,000	1717.616

   We now apply the collected data to verify that the asymptotic time complexity of bubble sort in its average case is $O(n^2)$ as the following. From the table above, we can see that when n grows with a factor of ten, the run time increases about one hundred times. That is quadratic because $100 = 10^2$.

Furthermore, the chart below illustrates the n-t data in log-log scale.



The line has a slope of 2, i.e. when log n increases one unit, log t increases about 2 units. It implies $\log t \approx 2 \log n$. Hence, $t = O(n^2)$.

Similarly, you should collect data for the best, and worst cases of bubble sort and verify the asymptotic time complexity experimentally.

3. For each of the selected algorithm, you should examine its best, worst, and average cases. Note that, for a given algorithm, some of the cases may have exactly the same asymptotic time complexity. You need to briefly explain the reason in your report.

What to Submit: Each group should submit:

1. A single copy of the documented source code; and
2. A single copy of the project report. *In your project report, you MUST specify individual contributions of each groupmate*. A sample outline below is for your reference.

- **Introduction**: A narrative overview: what is the project about? What are the objectives? How do you reach the objectives? What are the responsibilities and contributions of each group member?
- **The sorting algorithms to be studied**: Explain each of the selected sorting algorithms, with a flowchart and/or pseudocode, and derive its asymptotic time complexity in *best, worst*, and *average* cases with appropriate justifications.
- **Design of experiments:** algorithmic components, input data generation, collection of output data, integrating subtasks as a software solution.
- **Implementation and testing**: You do not need to include code segments. I will run and check your code anyway. However, you need to specify contribution of each member of your group. Every group member must be involved in implementation and testing.

- **Analyzing output data**: summarize output datasets in both tabular and graphical forms; compare the number of comparisons performed and time spent associated with data size. Verify if they match the results in theoretic analysis.
- **Summary**: Summarize your findings with conclusions and possible recommendations.

How it will be evaluated: I will read your report first and run your program with the rubric below to measure the expected learning outcomes 1 and 2 with the performance indicators:

1.a) Be able to specify the requirements of a complex computing problem through analyzing the problem; and further

1.b) Be able to identify mathematical and computational solutions to satisfy the requirements of the computing problem effectively.

2.a) Be able to implement algorithmic solutions into a software solution with an integrated development environment (IDLE) effectively; and further

2.b) Be able to evaluate and verify if the software solution satisfies the requirements of the computing problem.

		RUBRIC for SO 1			
		Unsatisfactory	**Developing**	**Satisfactory**	**Exemplary**
Performance Indicator		0 59	60 79	80 89	90 100
1.a	Analyze a complex computing problem (specifically, the testing suite for examining time complexity of sorting algorithms)	Unable to correctly analyze and specify critical requirements of the testing suite.	Be able to analyze and specify some requirements; however, the specified requirements may be incomplete and/or contain conceptual or logical mistakes.	Be able to analyze and specify critical requirements of the test suite mostly; and be able to present the specified requirements with appropriate justification mostly.	Be able to analyze and to specify all of the requirements of the test suite without any ambiguity; and be able to present the specified requirements very clearly with solid justification.
1.b	Apply principles of computing to identify solutions (specifically, for the testing suite)	Unable to identify available theoretical, computational, or algorithmic solutions to meet the specified requirements of testing suite.	Be able to identify available theoretical and computational solutions to meet some of the specified requirements; and/or be able to identify and construct appropriate input testing datasets for some of the best, worst, and average cases to verify asymptotic complexity.	Be able to identify theoretical and computational solutions to meet the specified requirements; and be able to identify and construct appropriate testing datasets for each of the best, worst, and average asymptotic complexity possibly with minor mistakes.	Be able to identify theoretical and computational solutions to meet all of the specified requirements precisely; and be able to identify and construct the most appropriate testing datasets for each of the best, worst, and average asymptotic complexity .

		RUBRIC for OS 2			
		Unsatisfactory	**Developing**	**Satisfactory**	**Exemplary**
Performance Indicator (R)		0 59	60 79	80 89	90 100
2.a	**Implement an appropriate solution for the design** (Time complexity of sorting algorithms)	Unable to implement algorithms that carry out the design correctly with appropriate data structures; The implementation is unorganized and/or poorly documented, hence, hard to follow; and/or The implementation contains bugs that cause unexpected termination.	Be able to implement algorithms that carry out the design correctly with appropriate data structures mostly; The implementation is somewhat organized with documentation, however, further clarifications are needed obviously; and/or The user interface and I/O management should be more user friendly.	Be able to implement algorithms that carry out all requirements of the design correctly with appropriate data structures and without runtime error; The implementation is organized with good documentation for others to follow; and/or The implementation is friendly with proper I/O interface.	Be able to implement algorithms that carry out all requirements of the design with effective/efficient data structures and software development tools available; The implementation is very well organized and easy to follow with clear documentation; and The implementation provides a very user friendly I/O interface.
2.b	**Evaluate if the solution meets the given set of requirements**	The software solution is not well evaluated; and/or The evaluations are not well justified.	The software solution is evaluated, however, not all of design objectives are verified; The evaluations do not consider best, worst cases; and/or The evaluations are not documented well with proper justifications.	The software solution is tested to verify if all design objectives are correctly met; Different scenarios are considered in solution evaluation; and The evaluations are documented with appropriate justifications mostly.	The software solution is very well tested and verified, such that, all design objectives are completely satisfied; Different scenarios are thoroughly considered in solution evaluation for robustness; and The evaluations are very well documented and justified.