COMP6008 DATA STRUCTURES AND ALGORITHMS

Summary

Title	Assessment 2 – Programming tasks: design of efficient algorithms
Type	Portfolio
Due Date	Friday 4 October 11:59 pm AEST (end of Week 6)
Length	NA
Weighting	50%
Academic Integrity (See below for limits of use where GenAI is permitted)	You **may use** GenAI tools to get some ideas or insight about particular problems in code. However, you **MUST NOT** generate a complete solution code or write a report. Please refer to the **Academic Integrity section** below to see what you can and cannot do with the GenAI tools.
Submission	A word document submitted to Turnitin using the provided template.
Unit Learning Outcomes	This assessment maps to the following ULOs: • ULO2: analyse the theoretical and practical time and space complexity of an algorithm • ULO3: demonstrate an understanding of algorithm design techniques for efficiency and/or clarity • ULO4: apply data structures and algorithms to solve a real-world problem

Rationale

Data structures and algorithms are fundamental skills for computer scientists. The ability to apply common data structures and design efficient algorithms to solve real-world problems often indicates a difference between computing and IT graduates. This assessment aims to assess students' skills in analysing algorithm complexity and designing efficient algorithms to solve problems. It addresses unit learning outcomes 2, 3, and 4 and relates to modules 4, 5, and 6.

Task Description

This assessment consists of a series of programming tasks, each requiring students to design efficient algorithms to solve well-defined problems.

These programming tasks are distributed across weeks 4, 5, and 6. The tutor supervises students to complete the coding parts of these tasks during each week's tutorial and workshop sessions, while students will complete the testing and reporting parts before the assessment submission due date. On-time completion of the coding parts is one of the marking criteria. By the end of each week's workshop session, the tutor checks each student's progress in completing the coding parts of the required tasks.

Task Instructions

The programming tasks cover the analysis of algorithm complexity and the design of efficient algorithms. For each task, you need to write solution code to solve the problem, provide a set of inputs and outputs to test the solution, attach screenshots of the tests, and include a summary of

comparison analysis if required. *If GenAI is used for the task, you must document how you use it and how it assists you in completing the tasks. Failing to do that will be subject to an academic integrity investigation*.

Module 4: Algorithm complexity (coding to be completed during Week Four's tutorial and workshop sessions)

Task 1 (4 marks): Write a program that prompts the user to enter a string and displays the maximum consecutive increasingly ordered substring. For example, if the input string is *"Welcome"*, then output should be *"Wel"*. Analyse the time complexity of your program.

Task 2 (4 marks): Write an O(n) program that prompts the user to enter a sequence of integers ending with 0 and finds the longest subsequence with the same number. For example, if the input sequence is 2 4 4 8 8 8 8 2 4 4 0, the output should be *"The longest same number sequence starts at index 3 with 4 values of 8.* Analyse why the time complexity of your program is O(n).

Task 3 (4 marks): Write an O(n) program that prompts the user to enter two strings and tests whether the second string is a substring of the first string. For example, if the first string is *"Welcome to Java"* and the second string is *"come"*, the output should be *"Matched at index 3"*. Analyse why the time complexity of your program is O(n). *Note: (1) Suppose the neighbouring characters in the strings are distinct, and (2) DO NOT use the indexOf method in the String class*.

Module 5: Efficient algorithms (coding to be completed during Week Five's tutorial and workshop sessions)

Task 4 (5 marks): Write a program that prompts the user to enter the size of an array. It first randomly fills an array of integers in {-1, 0, 1} and then sorts it with O(n) efficiency. For example, if the input is 10, a possible output of the unsorted array is 1 0 1 −1 0 1 −1 0 0 0, followed by the sorted array −1 −1 0 0 0 0 0 1 1 1. Analyse why the time complexity of your sorting program is O(n), considering that efficient sorting algorithms are at best O(nlogn), while inefficient algorithms can be $O(n^2)$.

Task 5 (8 marks): The bin packing problem is to pack objects of various weights into containers. Assume each container can hold a maximum of 10 pounds. The program uses an algorithm to place an object with the smallest weight into the first bin in which it would fit. Your program should prompt the user to enter the total number of objects and the weight of each object. The program displays the total number of containers needed to pack the objects and the contents of each container. Here is a sample run of the program:

```
Enter the number of objects: 6
Enter the weights of the objects: 7 5 2 3 5 8
Container 1 contains objects with weight 2 3 5
Container 2 contains objects with weight 5
Container 3 contains objects with weight 7
Container 4 contains objects with weight 8
```

Analyse the time and space complexity of your program. Discuss whether this program produces an optimal solution, that is, finding the minimum number of containers to pack the objects, and suggest a way to improve the solution. *Note: you are not required to implement the suggested solution*.

Module 6: Algorithms in practice (coding to be completed during Week Six's tutorial and workshop sessions)

Task 6 (5 marks): One way to improve the quicksort is to combine it with an insertion sort on lists that have a short length. If a list is longer than a threshold known as partition limit, execute quicksort; otherwise, execute insertion sort. We call it a hybrid sort algorithm. Implement the hybrid sort algorithm and design an experiment to find out the optimal partition limits for different list sizes.

Your program should create data randomly and print a table like this:

List size	Optimal partition limits
60,000	
120,000	
180,000	
240,000	
300,000	
360,000	
720,000	
1,000,000	

Hint: You can use the following code template to obtain the execution time:

```
long startTime = System.currentTimeMillis();
perform the task;
long endTime = System.currentTimeMillis();
long executionTime = endTime - startTime;
```

Task 7 (5 marks): When presidential candidate Barack Obama visited Google in 2007, Google CEO Eric Schmidt asked Obama the most efficient way to sort a million 32-bit integers (www.youtube.com/watch?v=k4RRi_ntQc8). Obama answered that the bubble sort would be the wrong way to go. Was he right? Write a program that obtains the execution times of bubble sort, selection sort, insertion sort, merge sort, quick sort, and hybrid sort (using optimal partition limit) for input sizes of 60,000, 120,000, 180,000, 240,000, 300,000, 360,000, 720,000, and 1,000,000. Run your program 100 times to get the average time for each sort.

Your program should create data randomly and print a table like this:

Input size	Bubble sort	Selection sort	Insertion sort	Merge sort	Quick sort	Hybrid sort
60,000						
120,000						
180,000						
240,000						
300,000						
360,000						
720,000						
1,000,000						

Resources

Use the following resources to support you when working on this assessment.

- Unit Modules 4-6 on Blackboard
- The reference books
- Java API documentation https://docs.oracle.com/en/java/javase/21/docs/api/index.html

Referencing Style Resource

Use Harvard style where applicable. Link: https://libguides.scu.edu.au/harvard

Task Submission

Submission of this assessment consists of two parts:

1. Coding parts: Completed in the respective tutorial and workshop sessions and checked by the tutor.
2. Portfolio: Submitted to the Blackboard unit site by the due date.

The portfolio must use the provided template, which includes code, testing inputs and outputs, screenshots of tests, answers to questions, analysis of experiment results, and reflection on the use of online resources, including GenAI, if any, for assisting in completing the tasks.

When you complete the assignment, you are required to submit the portfolio in Word DOCX file format using the **Turnitin** submission portal titled *Assessment 2 – Programming tasks: design of efficient algorithms* in the **Assessments Tasks and Submission** section on the Blackboard COMP6008 site. Only Microsoft Word documents submitted via the Turnitin portal on Blackboard will be accepted. The file will be named using the following convention: *LastNameFirstNameInitial_COMP6008_A2.docx, e.g., if a student's name is John Smith, the file name would be SmithJ_COMP6008_A2.docx*. ***Note: the code must be in text format and have appropriate indentation styles and comments. Screenshots of code are not accepted.***

Resubmit policy: This assessment is not eligible for a re-submit.

Academic Integrity

At Southern Cross University, academic integrity means behaving with the values of honesty, fairness, trustworthiness, courage, responsibility and respect in relation to academic work.

The Southern Cross University Academic Integrity Framework aims to develop a holistic, systematic and consistent approach to addressing academic integrity across the entire University. For more information, see: SCU Academic Integrity Framework

NOTE: **Academic Integrity breaches include** unacceptable use of generative artificial intelligence (GenAI) tools, the use of GenAI has not been appropriately acknowledged or is beyond the acceptable limit as defined in the Assessment, poor referencing, not identifying direct quotations correctly, close paraphrasing, plagiarism, recycling, misrepresentation, collusion, cheating, contract cheating, fabricating information.

At SCU the use of GenAI tools is acceptable, *unless it is beyond the acceptable limit as defined in the Assessment Item by the Unit Assessor*.

GenAI May be Used

Generative artificial intelligence (GenAI) tools, such as ChatGPT, **may be used** for this assessment task. If you use GenAI tools, you must use these ethically and acknowledge their use. To find out how to reference GenAI in your work, consult the referencing style for your unit via the Library referencing guides. If you are not sure how to, or how much, you can use GenAI tools in your studies, contact your Unit Assessor. If you use GenAI tools without acknowledgment, it may result in an academic integrity breach against you, as described in the Student Academic and Non-Academic Misconduct Rules, Section 3.

You **may use** Generative Artificial Intelligence (GenAI) tools, such as ChatGPT or Copilot, for this assessment task **to get some ideas or insight about particular problems in code**. It is similar when you try to find a snippet of code for doing a particular task in Stack Overflow. For example, **you must make your own effort to modify, refine, or improve the code to solve the assessment problems**. Think of it as a tool – a quick way to access information – or a (free) tutor to answer your questions. However, just as if you Googled something, you still need to evaluate the information to determine its accuracy and relevance. ***If you have used a GenAI tool in this assessment, you must document how you used it and how it assisted you in completing your assessment tasks***. ***Failing to do that will be subject to an academic integrity investigation***.

You **cannot use AI to generate a complete solution code or write a report**. You need to put your own effort into building the solution code for the assessments to demonstrate the required skills. Refer to assessment information in the Assessment Tasks and Submission area for details.

Special Consideration

Please refer to the Special Consideration section of Policy.
https://policies.scu.edu.au/document/view-current.php?id=140

Late Submissions & Penalties

Please refer to the Late Submission & Penalties section of Policy.
https://policies.scu.edu.au/view.current.php?id=00255

Grades & Feedback

Assessments that have been submitted by the due date will receive an SCU grade. Grades and feedback will be posted to the 'Grades and Feedback' section on the Blackboard unit site. Please allow 7 days for marks to be posted.

… continued on next page ...

Assessment Rubric

Note: The number of criteria used will vary depending on the complexity of the assessment task. Typically, 4–5 criteria are used.

Marking Criteria and % allocation	High Distinction (85–100%)	Distinction (75–84%)	Credit (65–74%)	Pass (50–64%)	Fail 0–49%
Criterion 1: Quality of solutions **ULOs**: 2, 3, 4 **Percentage**: 30%	The solutions completely and accurately solve the problems as specified.	The solutions mostly solve the problems as specified.	The solutions seem to solve the problems as specified.	The solutions seem to partially solve the problems as specified.	No or few solutions are provided.
Criterion 2: Testing of solutions **ULOs**: 2, 3, 4 **Percentage**: 20%	The solutions are thoroughly validated by tests.	The solutions are largely validated by tests.	The solutions are half validated by tests.	The solutions are sparsely validated by tests.	No or few tests are conducted.
Criterion 3: Complexity analysis of solutions **ULOs**: 2, 3, 4 **Percentage**: 10%	All the designed algorithms have sound complexity analysis.	75% of the designed algorithms have sound complexity analysis.	65% of the designed algorithms have sound complexity analysis.	Half of the designed algorithms have sound complexity analysis.	No or few designed algorithms have sound complexity analysis.
Criterion 4: Design and analysis of experiments **ULOs**: 2, 3, 4 **Percentage**: 10%	The experiments are solidly designed, and the results are thoroughly analysed.	The experiments are reasonably designed, and the results are reasonably analysed.	The experiments are designed, but the results are not well analysed.	The experiments are partially designed, and the results are incomplete.	No or few experiments are designed.
Criterion 5: On-time completion of solution codes **ULOs**: 2, 3, 4 **Percentage**: 30%	Complete all the required coding parts during the respective tutorial and workshop sessions.	Complete 2/3 of the required coding parts during the respective tutorial and workshop sessions.	Complete half of the required coding parts during the respective tutorial and workshop sessions.	Complete 1/3 of the required coding parts during the respective tutorial and workshop sessions.	Fail to complete any required coding part during the respective tutorial and workshop sessions.

Description of SCU Grades

High Distinction:

The student's performance, in addition to satisfying all of the basic learning requirements, demonstrates distinctive insight and ability in researching, analysing and applying relevant skills and concepts, and shows exceptional ability to synthesise, integrate and evaluate knowledge. The student's performance could be described as outstanding in relation to the learning requirements specified.

Distinction:

The student's performance, in addition to satisfying all of the basic learning requirements, demonstrates distinctive insight and ability in researching, analysing and applying relevant skills and concepts, and shows a well-developed ability to synthesise, integrate and evaluate knowledge. The student's performance could be described as distinguished in relation to the learning requirements specified.

Credit:

The student's performance, in addition to satisfying all of the basic learning requirements specified, demonstrates insight and ability in researching, analysing and applying relevant skills and concepts. The student's performance could be described as competent in relation to the learning requirements specified.

Pass:

The student's performance satisfies all of the basic learning requirements specified and provides a sound basis for proceeding to higher-level studies in the subject area. The student's performance could be described as satisfactory in relation to the learning requirements specified.

Fail:

The student's performance fails to satisfy the learning requirements specified.