Curtin University

School of Electrical Engineering, Computing & Mathematical Sciences

FINAL ASSESSMENT

End of Semester 2, 2022

COMP1002/COMP5008 Data Structures and Algorithms

This paper is for Curtin Bentley, SLIIT (CC), Mauritius and Miri students

This is an OPEN BOOK assessment

Assessment paper IS to be released to student

Examination Duration	24 hours	**Start:**	2:30pm 10th November WST (Perth) time
		Finish:	2:30pm 11th November (start/end time varies for CAP students)

Reading Time N/A

- Support will be available via Piazza for first four hours to answer questions

Total Marks 50

Supplied by the University

- Final Assessment paper
- Source material for Q1-6 in zip file – students must use the supplied code
- Piazza access to Tutors/Unit Coordinator

Supplied by the Student

- A programming environment
- All java code must be able to be compiled using javac *.java
- Python code must be able to run on the command line (e.g. python3 q3.py)

Instructions to Students

- **Attempt all questions.**
- Open book/computer, however, you must cite references.
- Keep all work within a directory: FinalAssessment_<Student_ID>
- Use given directory structure – each question in a separate directory
- Code for each task **must run** to be awarded any marks.
- Copied code will receive zero (0) marks
- Students must not work together or get help from other people
- **When complete:**
  - Read the Declaration of Originality – this will apply to your submission
  - Create a zip file of the Final Assessment directory (-r for recursive zip)
  - Submit to Assessment link on Blackboard before 2:30pm 11th November (Perth-time)
  - If you have problems uploading to Blackboard, send the zip to *comp1002@curtin.edu.au*
- **All submissions will be subjected to rigorous testing for plagiarism, collusion and any other forms of cheating. You must cite any and all design/java/python from any source, including your own work submitted for a different assessment.**
- **Assessment may include a follow-up demonstration or interview (viva)**

QUESTION ONE (Total: 10 marks): Recursion

a) **(2 marks)** Using the approach shown in the lectures, provide the token-by-token steps to convert infix to postfix for the following equations:

i. **30 + 12 * 3 + 2**
ii. **(30 + 12) * 3 + (4 + 5) * 7**

You should arrange the steps in table format as given below. Submit your work as a text file, word doc or scanned written work – you may include all Q1 answers in a single file.

Token	Postfix	Stack

b) **(2 marks)** Using the algorithm given in the lectures, show the successive calls in a **mergesort** of an eight-element array (A = the digits in your student ID*). Use **indenting** to show the depth of recursion based on the following methods:

- **mergesort(A)**
- **m_rec(A, l_index, r_index)**
- **merge(A, l_index, m_index, r_index)**

Submit your work as a text file, word doc or scanned written work.
**If your ID was 92345678, the values to sort would be [9,2,3,4,5,6,7,8]*

c) **(3 marks)** Using the algorithm given in the lectures, show the successive calls in a **quicksort** of the digits in your ID – sorted in **ascending** order*. Assume a **rightmost** pivot, and use **indenting** to show the depth of recursion based on the following methods:

- **quicksort(A)**
- **q_rec(A, l_index, r_index)**
- **partition(A, l_index, r_index, p_index)**

Submit your work as a text file, word doc or scanned written work.
**If your ID was 92345678, the values to sort would be [2,3,4,5,6,7,8,9]*

d) **(1 mark)** Based on your answers to questions **1b** and **1c** discuss sorting a 800 element array in terms of number of function calls and depth of recursion.

e) **(2 mark)** Implement a recursive greatest common denominator function and test harness in **Q1gcdTest.java/py**. Use your harness on the two four-digit numbers formed by splitting your studentID in half and testing in both orders (e.g. 92345678 -> (9234, 5678) and (5678, 9234)). Provide a diagram or list of the **call-stack** at the deepest level of recursion.

**** Don't forget to reference/cite sources, including your own code ****

QUESTION TWO (Total: 6 marks): Trees

a) **(6 marks)** Given the following list of numbers, manually generate the trees that would be created if the algorithms <u>shown in the lecture notes</u> were applied;

85, 5, 20, 15, 50, 55, 60, 65, 30, 35, 40, 55, 70

i) Binary Search Tree
ii) Red-Black Tree
iii) 2-3-4 Tree
iv) B-Tree (6 keys per node)

Note: *you can draw these on paper and submit a photo/scan, or write them up as text files or documents and add them into the zip file. Name them* ***Q2.****, *replacing * with the appropriate extension. Make sure they are clearly readable before submitting.*

In a text file **Q2discuss.txt**, reflect on the trees i-iv**)** in terms of how you would **delete** values

(1 mark per tree, 2 marks for discussion – if no working is shown, ½ mark per tree)

QUESTION THREE (Total: 9 marks): Hash Tables & Trees

a) **(4 marks)**. Given the supplied hash table code **Q3HashTable.java/py** write a program to read in the data from the supplied **6degrees.csv** file and store it to allow fast retrieval of movies, actors and movie roles. Your user interface should ask for the type of entry to retrieve and the string to search for. Along with the query result, it should give a query time measurement.

b) **(3 marks)**. Given the supplied binary search tree code **Q3BSTree.java/py** write a program to read in the data from the supplied **6degrees.csv** file and store it to allow fast retrieval of movies, actors and movie roles. Your user interface should ask for the type of entry to retrieve and the string to search for. Along with the query result, it should give a query time measurement.

(we recommend code reuse between 3b and 3c)

c) **(2 marks)**. Based on the results from 3b and 3c, discuss the theoretical and actual performance of the two approaches. Provide data and complexity theory to support your discussions.

Note*: you can use the PracExamException supplied for any exceptions.*

QUESTION FOUR (Total: 7 marks): DSA in Practice

a) **(7 marks)** Based on the **Priority Queue** Abstract Data Type:

- Create a Traceability Matrix of the functionality you will be testing for the Priority Queue (column 1) **(1 mark)**
- Code a test harness **Q4PriorityTest.py/java** using a **built-in Priority Queue implementation** from your Python/Java. The harness should cover the functionality listed in your Traceability Matrix **(2 marks)**
- Populate the Traceability Matrix referencing the built-in **Priority Queue** functionality (column 2) and the test cases in your test harness and their outcomes (column 3) **(2 marks)**
- You should test for error states as well as normal performance, using exception handling as required**.** **(2 marks)**

Traceability Matrix

Functionality	**Implementation**	**Tests**

**** Don't forget to reference/cite sources, including your own code ****

QUESTION FIVE (Total: 13 marks): Graphs

***Note**: Use/modify the provided **Q5Graph.java/py** code throughout this question*

a) **(2 marks)**. You will be writing code to represent the connections between movies, actors and movie roles. After familiarising yourself with the data in **6degrees.csv**, use diagrams and descriptions to explain how you will represent the data as a graph.

(read ahead in the questions below to see the functionality required)

b) **(4 marks)**. Modify **Q5Graph.java/py** as needed to represent the data.

- Code changes for Graph **(1 mark)**
- Include three display methods to show **movies**, **actors** and **roles**. **(1 mark)**
- Create sample data based on the provided file – this should not be a large file (~20 entries)
- Provide code in **GraphTest.java/py** to show you've thoroughly tested the functionality of the code. **(1 mark)**
- Document the code **(1 mark)**

c) **(4 marks)**. Write code for the methods:

- **displayActorsMovies(name)** to display a list of all movies that an actor has appeared in **(1 mark)**
- **displayMovieActors(name)** to display a list of all actors in a given movie **(1 mark)**
- Provide code in **GraphTest.java/py** to show you've thoroughly tested the functionality of the code. **(1 mark)**
- Document the code **(1 mark)**

d) **(3 marks)**. Write code for the method **displayCostars(name)** to generate the **co-stars** of a particular actor. It should output each Movie name with co-stars indented.

The format must be:

```
Costars for Colin Firth

Pride and Prejudice
   Actor 1
   Actor 2
Bridget Jones' Diary
   Actor 1
   Actor 2
```

Provide code in **GraphTest.java/py** to show you have thoroughly tested the functionality of the method.

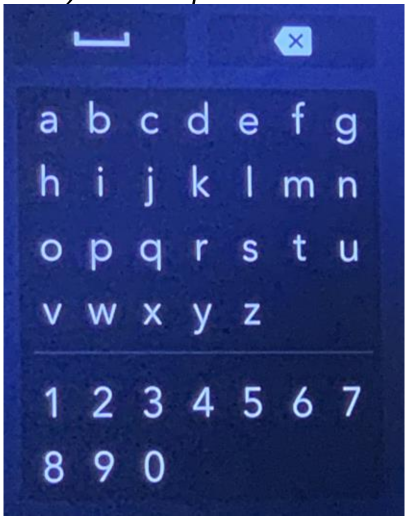
**** Don't forget to reference/cite sources, including your own code ****

QUESTION SIX (Total: 5 marks): Assignment

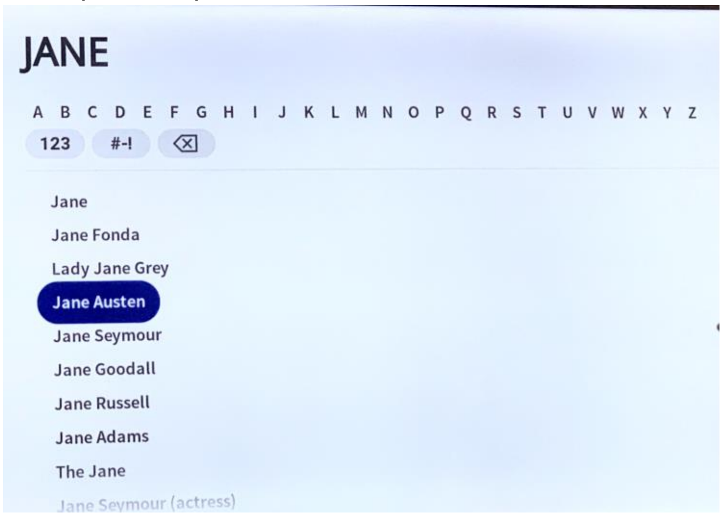
Note : you do not have to code/implement this question

a) **(2 marks)**. **Describe** how you would extend your assignment to represent and include the following two keyboards

1) *No wrap*

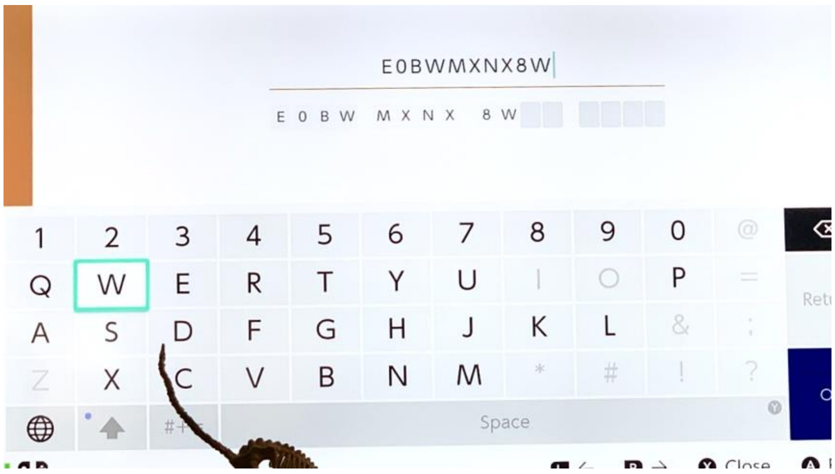


2) *No wrap*



b) **(2 marks)**. **Discuss** the respective performance you might expect for the keyboards in Question 6a. Provide test data that would help demonstrate the expected performance.

c) **(1 mark)**. The keyboard below is enhanced to optimise data entry based on the specific data being entered. **Discuss** how this strategy could (or couldn't) be supported by your approach to the assignment.



(ignore the tail of the Halloween cat 😊)

**** Don't forget to reference/cite sources, including your own code ****

END OF ASSESSMENT

**** Don't forget to reference/cite sources, including your own code ****