Curtin University

School of Electrical Engineering, Computing & Mathematical Sciences

FINAL ASSESSMENT

End of Semester 2, 2021

COMP1002/COMP5008 Data Structures and Algorithms

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

This is an OPEN BOOK assessment

Assessment paper IS to be released to student

Examination Duration 24 hours **Start:** 12:00pm 13th June WST (Perth) time
Finish: 12:00pm 14th June
(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 Q1-5 in zip file – students must use the supplied code
- Collaborate and 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>, using given directory structure
- 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 12:00pm 14th June (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. **100 + 2 * 3 + 2**
ii. **(100 + 2) * 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.

Token	Postfix	Stack

b) **(3 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, leftindex, rightindex)**
- **merge(A, leftindex, middleindex, rightindex)**

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) **(4 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 **leftmost** pivot, and use **indenting** to show the depth of recursion based on the following methods:

- **quicksort(A)**
- **qsort_rec(A, leftindex, rightindex)**
- **dopart(A, leftindex, rightindex, pivotindex)**

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 100 element array in terms of number of function calls and depth of recursion.

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

QUESTION TWO (Total: 10 marks): Trees

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

100, 15, 20, 25, 50, 55, 60, 65, 30, 35, 40, 45, 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)

b) **(1 mark)**. Given the supplied code **Q2BSTree.java/py** for a Binary Search Tree and associated TreeNode classes, extend the code to add a **height** variable to each node, which will be initially set to **-1**. Provide code in **TreeTest.java/py** to test its functionality.

Note: *delete functionality is not required*

c) (3 marks). Write the method **genTreeHeights()** to update the height values of all the nodes in the tree. Update the other tree code as required so that the heights are always up to date.

Provide code in **TreeTest.java/py** to show you've thoroughly tested the functionality of the method.

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

QUESTION THREE (Total: 10 marks): Heaps

a) **(2 marks)** Using the digits in your student ID as a list of numbers, manually generate the **max heap** and the **min heap** that would be created if the algorithm shown in the lecture notes is applied.

If your ID was 92345678, the values to use would be [9,2,3,4,5,6,7,8]

Note: *show your working in multiple steps. 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 **Q3.***, replacing * with the appropriate extension. Make sure they are clearly readable before submitting.*

b) **(3 marks)**. Given the supplied max heap code to **Q3Heap.java/py** to convert it to use a dynamic tree with objects (as in the Binary Search Tree) instead of an array. Indicate any changes made using inline comments. Create a test harness **HeapTest.java/py** to demonstrate that it is working.

Note: *if you can't get this to work, you can still do the following parts…*

c) **(1 mark)**. Modify the **Q3Heap** code to throw appropriate exceptions in the **add** and **remove** methods. Add code to **HeapTest.java/py** to show the exceptions are thrown as and when expected. Put comments in the test harness to explain your changes.

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

d) **(3 marks)**. Extend **Q3Heap.java/py** to read in the priority queue data from the provided file. The data has a priority (movie rating) and a value, both of which need to be read in and stored in the heap. Your tests should include printing out the contents (priority and value) of the priority queue after each element is added. It should then do repeated removals, again printing the priority queue, until the queue is empty.

QUESTION FOUR (Total: 8 marks): DSA in Practice

a) **(8 marks)** Based on the Stack abstract data type:

- Create a Traceability Matrix of the functionality you will be testing for a stack
- Code a test harness Q4Stack_Test.py/java using a **built-in Stack implementation** from your language, designed to cover the functionality of the stack
- Populate the Traceability Matrix referencing the built-in Stack and the test cases in your test harness (and their results)
- Remember to test for error states as well as normal performance, using exception handling as required**.** **(2 marks per dot-point)**

ity	Implementation	Test

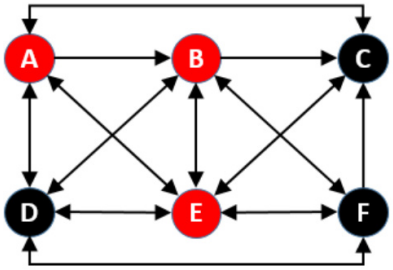
**** Don't forget to reference/cite sources, including your own code ****

QUESTION FIVE (Total: 12 marks): Graphs

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

a) **(4 marks)**. Write code to read in the graph data from the provided file. An example graph is on the right.

- The graph is **directed**
- Nodes have **colours**
- Edges have **weights** (not shown)



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

***Note:** if you can't get colours or file I/O to work, you can still do the other parts of the questions…*

b) **(4 marks)**. Write code for the method **displayColourMatrix(colour)** to generate the **adjacency matrix** representation of the subgraph with the matching node colour, or all colours, and indicating the **weights** of the edges in the matrix.

The format must be:

```
Weight matrix for graph[red] is:

     A  B  E
-----------
A | 0  4  5
B | 0  0  3
E | 5  3  0
```

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

c) **(4 marks)**. Write code for the method **displayColourList(colour)** to generate the **adjacency list** representation of the subgraph with nodes matching the selected colour, or all colours. Each link should have the weight in brackets

The format must be:

```
Adjacency list for graph[red] is:

A | B(4), E(5)
B | E(3)
E | A(5), B(3)
```

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

END OF ASSESSMENT

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