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 and Miri students

This is an OPEN BOOK assessment

Assessment paper IS to be released to student

Examination Duration 24 hours **Start:** 12:00pm 3rd November WST (Perth) time
Finish: 12:00pm 4th November
(start/end time varies for CAP students)

Reading Time N/A

- Support will be available via Collaborate/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:**
  - Sign Declaration of Originality and save into Final Assessment directory
  - Create a zip file of the Final Assessment directory (-r for recursive zip)
  - Submit to Assessment link on Blackboard before 12:00pm 4th November (Perth-time)
  - If there are problems uploading to Blackboard, email yourself a copy (Curtin email)
- **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) **(3 marks)** Using the algorithm given in the lectures, provide the token by token steps to convert infix to postfix for the following equations:

i. **10 - 4 / 3 + 7**
ii. **(10 - 4) / 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 a 7 element array (A). 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.

c) **(4 marks)** Using the algorithm given in the lectures, show the successive calls in a **quicksort** of a 7 element array (A). Assume a **leftmost** pivot with **ascending** data, 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.

d) **(2 marks)** 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: 12 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;

10, 15, 50, 55, 60, 65, 30, 35, 20, 40, 25, 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)

b) **(2 marks)**. Given the supplied code for the **Binary Search Tree** and associated TreeNode classes, extend the code to add a **colour** to each node, which will be initially set to **black**. Provide code in **TreeTest.java/py** to test its functionality.

c) **(4 marks)**. Write the method **colourTree()** to update the colour values of all the nodes in the tree. The colouring rules are:

1. Nodes with two children are to be black
2. Nodes with one child are brown
3. Leaf nodes are orange (if in the first two levels) or red (levels 3+)

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: 8 marks): Heaps

a) **(2 marks)** Given the following list of numbers, manually generate the **max heap** that would be created if the algorithm shown in the lecture notes is applied:

50, 30, 100, 40, 60, 50, 80, 70, 50, 90

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.*

a) **(2 marks)**. Copy the given max heap code to **FA_MinHeap.java/py** to implement a **min** heap. Indicate any changes made using inline comments. Create a test harness **MinHeapTest.java/py** to demonstrate that it is working.

b) **(2 marks)**. Modify the **FA_MinHeap** code to throw appropriate exceptions in the **add** and **remove** methods. Add code to **MinHeapTest.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.

c) **(2 marks)**. Extend **FA_MinHeap.java/py** to read in the priority queue data from the provided file. The data has a priority 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: 6 marks): DSA in Practice

a) **(2 marks)**. Modify the given **FA_HeapTest.java/py** to use built-in datatype(s) for Priority Queues in Java/Python. If an operation doesn't have an equivalent, comment it out and note the issue.

b) **(4 marks)** Copy **Sorts.java/Sorts.py** and the **SortsTestHarness** (from your Pracs or the Practicals area on Blackboard) to the Question 4 directory.

We will be investigating the following sorts. Implement them if you do not already have them:

- Insertion Sort
- Quicksort using Median of 3 pivot strategy
- Radix sort using Most Significant Digit (your choice of base)
- Built-in sort for your language

Edit the **text** file (Q4_SortsAnalysis.txt) to explain your method for analysing the performance for the listed Sorts, and your interpretation of the results. This is what we will assess – one mark per sort. There should be a table plus a paragraph or two per sort, relating the performance to the theoretical performance for each sort.

Q4_SortsAnalysis.txt

```
wwSort
Theoretical Performance
- Best Case: <e.g. ascending>
- Worst Case:
- Average Case:

Data Size  | Average Case
1000       |
10000      |
100000     |

Discussion :

<repeat for each sort>
```

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

QUESTION FIVE (Total: 12 marks): Graphs

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

a) **(5 marks)**. Write code to read in the graph data from the provided file. The graph should be **directed**, and the **weights** need to be read into the graph.

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

b) **(3 marks)**. Write code for the method **displayAsWeightMatrix()** to generate the **adjacency matrix** representation of the graph – showing the **weights** of the edges.

The format must be:

```
Weight matrix for graph is:

    A  B  C
-----------
A - 0  3  1
B - 7  0  5
C - 2  0  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 **displayAsList()** to generate the **adjacency list** representation of the graph.

The format must be:

```
Adjacency list for graph is:

A - B C
B - A C
C - A
```

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 ****