Curtin University

School of Electrical Engineering, Computing & Mathematical Sciences

FINAL ASSESSMENT

End of Semester 2, 2020

COMP1002/COMP5008 Data Structures and Algorithms

This paper is for Curtin Bentley, Mauritius and Miri students

This is an OPEN BOOK assessment

Assessment paper IS to be released to student

Examination Duration 24 hours **Start:** 1:00pm 17th November WST (Perth) time
Finish: 1:00pm 18th 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, Piazza and email 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 1:00pm 18th 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) **(4 marks)** We will be using the following algorithms to give a performance profile and discussion of each. Implement them if you do not already have them:

- Iterative Factorial
- Recursive Factorial
- Iterative Fibonacci
- Recursive Fibonacci

You need to write a test harness (FA_RecursionHarness.java/py) to call the various algorithms and output the timing. You might use the SortsTestHarness as an inspiration.

Edit the **text** file (Q1_RecursionAnalysis.txt) to discuss performance for the recursive and iterative algorithms. You must use the input values as given, but can **extend** on them (`10 | 20 | 30 | 40 | 50 | 60 | 70  <- input values)`

This is what we will assess – one mark per algorithm. Discussion should be a few paragraphs per algorithm, explaining your reasoning for the different performance of each.

Q1_RecursionAnalysis.txt

```
Algorithm  | 10 | 20 | 30 | 40 | 50 | 60 | 70<- input values
It. Fact   |    |    |    |    |    |    |
Rec. Fact  |    |    |    |    |    |    |
Iter. Fib  |    |    |    |    |    |    |
Rec. Fib   |    |    |    |    |    |    |


Recursive Discussion

Input data + performance

Discussion

Fibonacci Discussion

Input data + performance

Discussion
```

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

b) **(2 marks)**. Given the supplied code for the **Binary Search Tree** and associated TreeNode classes, write the method **printEvenValues()** to print the nodes with **odd values** in the tree. Provide code in **TreeTest.java/py** to test its functionality (Hint: you will need to traverse the tree).

c) **(4 marks)**. Write the method **printEvenLevels()** to print all values in the odd levels of the tree. (Hint: you can base this on a traversal of the tree, and can use other code supplied with this Assessment).

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

QUESTION TWO (Total: 10 marks): Evaluating Algorithms

a) **(4 marks)** Copy **Sorts.java/Sorts.py** and the **SortsTestHarness** (from your Pracs or the Practicals area on Blackboard) to the Question2 directory.

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

- Insertion Sort
- ShellSort
- Two versions of Quicksort
  - *i.* Pivot strategy 1 is Rightmost
  - *ii.* Pivot strategy 2 is Median of 3

Edit the **text** file (Q2_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.

Q2_SortsAnalysis.txt

Selection Sort

```
Data Size  | Best Case | Worst Case | Average Case
           |           |            |
           |           |            |
           |           |            |
           |           |            |
```

Discussion :

ShellSort

```
Data Size  | Best Case | Worst Case | Average Case
           |           |            |
           |           |            |
           |           |            |
           |           |            |
```

etc.

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

b) **(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;

00,11,22,33,44,88,99,100,44,55,66,77

i) Binary Search Tree
ii) Red-Black Tree
ii) 2-3-4 Tree
iii) 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 from **b)** in terms of:

I. the heights of the resultant trees – how do they compare for the same input values?
II. Compare the understandability of the algorithms, which would be easier to implement?

(1 mark per tree/discussion point)

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

QUESTION THREE (Total: 12 marks): Heaps

a) **(3 marks)** Modify **FA_MaxHeap.java/py** to store the priority and value. Provide code in **MaxHeapTest.java/py** to show you've tested the functionality of the Heap.

b) **(3 marks)**. Modify the **FA_MaxHeap** code to throw appropriate exceptions in the **add** and **remove** methods. Add code to **MaxHeapTest.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) **(3 marks)**. Extend **FA_MaxHeap.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.

d) **(3 marks)**. Copy the given heap code to **FA_MinHeap.java/py** to implement a **min** heap. Indicate any changes made using inline comments. Copy the previous test code to **MinHeapTest.java/py** code to demonstrate that it is working. Put comments in the test harness to explain your changes.

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

QUESTION FOUR (Total: 6 marks): Stacks/Queues and Built-ins

a) **(3 marks)**. Modify the given **FA_StackTest.java/py** to use built-in datatype(s) for Stacks in Java/Python. If an operation doesn't have an equivalent, comment it out and note the issue.

b) **(3 marks)**. Modify the given **FA_HashTableTest.java/py** to use built-in datatype(s) for Hash Tables in Java/Python. If an operation doesn't have an equivalent, comment it out and note the issue.

**** 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 tested the functionality of the method.

b) **(4 marks)**. Write code for the method **displayAsMatrix()** to generate the **adjacency matrix** representation of the graph.

The format must be:

```
Adjacency matrix for graph is:

     A  B  C
===========
A | 0  1  1
B | 1  0  1
C | 1  0  0
```

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

c) **(3 marks)**. Write code for the method **displayWeightMatrix()** to generate the **adjacency matrix** representation of the graph – this time 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 tested the functionality of the method.

END OF ASSESSMENT

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