School of Electrical Engineering, Computing & Mathematical Sciences

FINAL ASSESSMENT

End of Semester 1, 2021

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:** 12:00pm 8th June WST (Perth) time
Finish: 12:00pm 9th 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

- Zip file including Final Assessment paper and source material for Q1-5 in zip file
  - **Students must use the supplied code**
- 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 through Assessment link on Blackboard before 12:00pm 9th June (Perth-time)
  - If you have problems uploading to Blackboard, email 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: 13 marks): Evaluating Algorithms

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 Fibonacci
- Recursive Fibonacci
- Recursive Fib-factorial
    - fibfact(n) = fibfact (n-1) * fibfact (n-2)
    - fibfact (0) = 1, fibfact(1) = 1
- Recursive FibOf3
    - fib3(n) = fib3(n) + fib3(n-1) + fib3(n-2)
    - fib3(0) = 1 , fib3(1) = 1, fib3(2) = 1

You need to write a test harness (Q1a.java/py) to call the various algorithms and output the timing. You might use the SortsTestHarness as an inspiration.

Create a **text** file as shown below (Q1a.txt) to discuss performance for the recursive and iterative algorithms. You must use the input values as given, but can **extend** on them (`5 | 10 | 20 | 30 | 40 <- input values)`

The discussion 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.

Q1a.txt

```
Algorithm    |  5 | 10 | 20 | 30 | 40 <- input values
Rec. Fib     |    |    |    |    |
Rec. Fib     |    |    |    |    |
Rec. Fibfact |    |    |    |    |
Rec. Fib3    |    |    |    |    |


<Discussion of each set of results>
```

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

b) **(5 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:

- BubbleSort
- ShellSort (Lecture 11)
- MergeSort
- Two versions of Quicksort
    - *i.* Pivot strategy 1 is Leftmost
    - *ii.* Pivot strategy 2 is Median of 3

Edit the **text** file (Q1b.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.

Note 1: Include a table plus a paragraph or two per sort, relating the performance to the theoretical performance for each sort. Indicate which input options you've chosen to give best/worst/average performance. As a guide, data sizes should be in the range 100-100,000 and there should be three or more runs.

Note 2: You can do multiple runs with one command:

```
python3 SortsTestHarness.py 1000 id ia sd sa

java SortsTestHarness 1000 id ia sd sa
```

Q1b.txt

```
<??> Sort

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

Discussion :

<??> Sort

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

etc.
```

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

c) **(4 marks)** Hashing algorithms affect the performance of hashing according to 4 properties – covered in the lecture notes. Property 4 – distributes keys evenly - is the most challenging.

- Implement two hashing algorithms - **badhash(key)** with poor distribution and **goodhash(key)** must have good distribution.
- Write a test harness (Q1c.java/py) to demonstrate the distribution of keys by the two functions (2 marks)
- Create a text file or Word doc Q1c.txt/doc (plotting can be used) to discuss your results (2 marks)

Note1: You do not need a hash table to answer this question.
Note2: The keys can be integers or strings – the choice is yours.

QUESTION TWO (Total: 13 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;

15, 18, 21, 3, 6, 9, 12, 24, 27, 30, 33, 36

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

Note1: *You must show some of your working – at least 2 steps and a final per tree*
Note2: *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 **Q2a.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)

b) **(3 marks)**. Given the supplied code for the **Binary Search Tree** and associated TreeNode classes, write the method **printThreeValues()** to print the minimum, root and maximum values in the tree.

Provide code in **Q2TreeTest.java/py** to thoroughly test its functionality

c) **(4 marks)**. Write the method **printTwoLevels()** to print all values in the top two levels of the tree – below the root (don't include the root). (Hint: you can base this on a traversal of the tree, and can use other code supplied with this Assessment).

Provide code in **Q2TreeTest.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) **(3 marks)**. Modify the **Q3MaxHeap** code to throw appropriate exceptions in the **add** and **remove** methods. Add code to **Q3MaxHeapTest.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.

b) **(3 marks)**. Extend **Q3MaxHeapTest.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.

c) **(2 marks)**. Copy the given heap code to **Q3MinHeap.java/py** to implement a **min** heap. Indicate any changes made using inline comments. Copy the previous test code to **Q3MinHeapTest.java/py** code to demonstrate that it is working. Put comments in the test harness to explain your changes.

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

a) **(3 marks)**. Modify the given **Q4StackTest.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 **Q4HashTableTest.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.

Note: Stack and queue code is provided in “for_info_only” – they are only supplied to demonstrate the expected functioning of the test harness. You should only be editing the test harnesses.

QUESTION FIVE (Total: 10 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. The graph should be **undirected**, and the **weights** need to be read into the graph.

Provide code in **Q5GraphTest.java/py** to show you've tested the functionality of the method.

b) **(3 marks)**. Write code for the method **displayWeightsList()** to generate the **adjacency list** representation of the graph, including each weight in brackets.

The format must be:

```
Weighted adjacency list for graph is:

    A - B:3 > C:1
    B - A:3 > C:2
    C - A:1 > B:2
```

Provide code in **Q5GraphTest.java/py** to show you've tested the functionality of the method.

c) **(3 marks)**. Write code for the method **displayWeightsMatrix()** 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 - 3  0  2
    C - 1  2  0
```

Provide code in **Q5GraphTest.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 ****