The University of Nottingham

SCHOOL OF COMPUTER SCIENCE

A LEVEL 2 MODULE, FULL YEAR, 2020-2021

ALGORITHMS, CORRECTNESS AND EFFICIENCY

Deadline for submission: see the Moodle submission box

Open-book examination.

Answer ALL SIX questions

Suggested time to complete the paper ~2 hours.

This open-book examination will be marked out of 100.
Marks available for questions, and sections of questions, are shown in parentheses.

You may write/draw by hand your answers on paper and then scan them to a PDF file, or you may type/draw your answers into electronic form directly and generate a PDF file. Guidance on scanning can be found through the Faculty of Science Moodle Page Guidance for Remote Learning. Your solutions should include complete explanations and should be based on the material covered in the module. Make sure your PDF file is easily readable and does not require magnification. Make sure that each page is in the correct orientation. Text/drawing which is not in focus or is not legible for any other reason will be ignored.

Submit your answers containing all the work you wish to have marked as a single PDF file on the module's Moodle page.

Start the answer for each question on a new page within the PDF

Use the standard naming convention for your document: "[Student ID]-COMP2009-2020-21"
Write your student ID number at the top of each page of your answers.

Although you may use any notes or resources you wish to help you complete this open-book examination, the academic misconduct policies that apply to your coursework also apply here. You must be careful to avoid plagiarism, collusion, or false authorship. Please familiarise yourself with the Guidance on Academic Integrity in Alternative Assessments, which is available on the Faculty of Science Moodle Page Guidance for Remote Learning. The penalties for academic misconduct are severe.

Staff are not permitted to answer assessment or teaching queries during the period in which your examination is live. If you spot what you think may be an error on the exam paper, note this in your submission but answer the question as written.

1) DELETED AS NOT RELEVANT TO COMP2054

2) DELETED AS NOT RELEVANT TO COMP2054

3) DELETED AS NOT RELEVANT TO COMP2054

4) This question is concerned with the 'big-Oh' family. (20 marks total)

(a) The usual definition of 'big-Omega' is that

```
f(n) is Ω( g(n) ) if and only if
there exists c > 0 and n0, such that,
forall n ≥ n0
   f(n) ≥ c g(n)
```

Carefully explain, in your own words, the reasoning and motivation that justify this definition.

Also, explain, with an example why the following (incorrect) definition, would not be suitable or useful:

```
f(n) is Ω( g(n) ) if and only if
there exists n0, such that,
forall n ≥ n0,
there exists c > 0 such that,
   f(n)  ≥  c g(n)
```

(5 marks)

(b) Give the definitions of big-Theta and little-oh by completing each of the following, and for each one you should also briefly explain, in your own words, the reasoning and motivation that justify the definitions:

i) 'big-Omega': f(n) is Θ(g(n)) ….

ii) 'little-oh': f(n) is o(g(n)) …. (4 marks)

(c) Consider the function defined by the following code

```
int f(int n) {
    int r = 0;
    while ( n > 5 ) {
        n = n – 5;
        r++;
    }
    return r;
}
```

From the definitions:

i) Prove or disprove that f(n) is Θ (n) (`Theta of n').

ii) Prove or disprove that f(n) is o (n) (`little-oh of n')

In each of these cases, you should also explain why the result is reasonable and matches the motivations underlying the appropriate definition.

(5 marks)

(d) Consider the following recurrence relation

$T(1) = 1$

$T(n) = 4\,T(n/3) + 2$

i) Give the exact solution, with an explanation of how you found it, and prove it correct by using induction. Also express your answer in terms of n, and also to give the Big-Theta behaviour of your answer.

ii) Use the Master Theorem to give the Big-Theta behaviour. Explain your answer.

(6 marks)

5) This question concerns sorting algorithms and Binary Search Trees (15 marks total)

(a) Suppose that we consider an array A of n distinct integers that is “d-sorted” which will mean that every entry is guaranteed to be no more than a distance d from the position it would take if the array were sorted. For example, for a 1-sorted array A, the integer at index k, will move to one of {k-1,k,k+1} when the array is sorted. For example [1 2 4 3 5] is 1-sorted, and [2 1 5 3 4] is 2-sorted as the ‘5’ needs to move 2 spaces.

Give algorithms, using pseudo-code, that can take as inputs the value d, and a d-sorted array A, and that gives as output a sorted array, and that are based upon each of:

i) Insertion sort

ii) Bubble sort

For fixed but arbitrary constant d, the algorithms should run in time O(n).
E.g. for d=1 they should be O(n), for d=2 they should be O(n), etc.
You should justify why they are correct, and also justify that they are O(n).

Give a small example of their working with the 2-sorted array [2 1 5 3 4].

(6 marks)

(b) Describe and explain the conditions for a Binary tree to be a Binary search tree. Then explain the operation for deleting an element, justify why it is correct, and give its complexity as a function of the size and/or height of the tree.

(4 marks)

(c) Give a method that runs in O(n log n) and that will allow insertion of n elements from an array into a BST in such a way that the height of the tree is guaranteed to be O(log n).

(5 marks)

6) This question concerns graph algorithms. (15 marks total)

(a) Give a brief description and explanation of Prim's algorithm to find a Minimum Spanning Tree (MST) of a connected undirected graph G = (V,E) with weights on the edges.

Then give an argument for why the algorithm is correct, that is, why it will always return a MST.

(5 marks)

(b) Suppose that in the graph G all the edges have distinct weights – that is, no two edge have the same weight. Then it is known that the resulting MST is unique, that is, G cannot have two different MSTs.

Is it then true that the minimum weight edge within G must always appear in the unique MST? Justify your answer.

(4 marks)

(c) Consider the Floyd Warshall (FW) algorithm on an undirected graph but that is known to consist of two disconnected components V1 and V2 (there is no path from any node in V1 to any node in V2). Also suppose that the components V1 and V2 are of roughly equal size, though you are not told in advance which nodes of G are in which component. How would you exploit this knowledge to reduce the space and time usage of the FW algorithm? Explain and justify your answer.

(6 marks)