Semester 2 Examinations, 2018-2019

Exam Code(s)	2BCT1, 2BLE1, 2BP1
Exam(s)	Second BSc in Computer Science & Information Technology Second BE in Electrical & Electronic Engineering Second BE in Electronic & Computer Engineering
Module Code(s)	CT2109
Module(s)	Object Oriented Programming: Data Structures and Algorithms
Paper No.	1
External Examiner(s)	Dr. Jacob Howe
Internal Examiner(s)	Prof. Michael Madden *Dr. Frank Glavin
Instructions*:*	Answer any three questions. All questions carry equal marks.
Duration	2 hours
No. of Pages	5
Discipline(s)	Information Technology
Course Co-ordinator(s)	Colm O'Riordan
Requirements	None

[PTO]

Question 1

a) Define the following concepts and provide examples for each.
   (i) Recursion
   (ii) Dynamic Programming
   (iii) Big O Notation

[6 Marks]

b) The *Fibonacci Sequence* is the series of numbers:

   0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

   The next number is found by adding up the two numbers before it. This can be implemented recursively, and the recursive calculation is defined for positive integers as f(n) where:

   $f(0) = 0; f(1) = 1; f(n) = f(n-1) + f(n-2)$

   Using appropriate diagrams and/or code snippets, demonstrate how the recursive calculation of the Fibonacci sequence can be *improved by applying the dynamic programming* approach.

[5 Marks]

c) An algorithm's performance may be measured experimentally or theoretically.
   (i) Describe in detail how theoretical algorithm analysis is performed.
   (ii) Describe in detail how you would perform an experimental comparison of two different algorithms for one task.
   (iii) What, in your opinion, are the advantages of each approach?

[9 Marks]

d) Provide a definition for P and NP problems and give an example of each. Explain what is meant by the "*P versus NP*" problem.

[5 Marks]

[PTO]

Question 2

a) Describe how the Radix Sort algorithm operates, using appropriate pseudocode, Java code and/or diagrams to illustrate your description.

[6 Marks]

b) Starting with the following array of two-digit integers, demonstrate how the Radix Sort algorithm that you described above operates, by working through all iterations/steps until the array is sorted:

[12, 98, 34, 67, 54, 23, 56, 45]

[4 Marks]

c) Provide algorithms for both *Binary Search* and *Sequential Search*, including details of any requirements that are imposed on their inputs. Demonstrate how the binary search algorithm would work for the following list of elements when the search item is 16.

[1, 4, 7, 8, 13, 15, 16]

[7 Marks]

d) With reference to your algorithms in Part (c) above, derive O-notation expressions for the efficiency of both Sequential and Binary Search.
Note: You should explicitly show your reasoning and not just provide the O-notation.

[4 Marks]

e) Explain in detail the distinctions between ‘big Omega’ and ‘big Theta’ complexity analyses.

[4 Marks]

[PTO]

Question 3

a) Define what is meant by a *binary tree*. Illustrate how it can be used to represent arithmetic expressions. Provide an example of another useful application of a binary tree.

[6 Marks]

b) Using appropriate diagrams, differentiate between *Pre-Order Traversal*, *Post-Order Traversal,* and *In-Order traversal* in the context of traversing a binary tree.

[6 Marks]

c) In the context of compression algorithms, describe what is meant by *Huffman Encoding*. Outline the steps (using both a text explanation and diagrams) for constructing a *Huffman Tree*.

[8 Marks]

d) In the context of AVL trees, and using the dataset below, illustrate how to perform the following operations:
   (i) Inserting the data into an AVL tree (note: clearly identify and describe all rotations)
   (ii) Searching for a given data item within the AVL tree.

The dataset is: 60, 50, 20, 80, 90, 30

[5 Marks]

[PTO]

Question 4

a) Define what is meant by an *Abstract Data Type*. Write code to define a Java **interface** corresponding to a Stack ADT and provide a short description for each of the operations.

   Operations should include: add item to the stack, remove item from the stack, reading the top element of the stack, checking if it is full, checking if it is empty.

[5 Marks]

b) Using the interface from Part (a) above, write a basic Java implementation of the Stack interface. An array should be used for the underlying storage.

[5 Marks]

c) Write a class called *StackTest* for testing your implementation from Part (b). You should create an instance of your ArrayStack class and test the functionality of each of the methods.

[3 Marks]

d) For a regular array implementation of a Queue ADT, items must be "shuffled" towards the front after a dequeue. Is there a way of implementing a Queue, as an array, while avoiding these shuffles? Explain your answer and use diagrams as appropriate.

[3 Marks]

e) What are the key differences between the *singly* and *doubly* linked list data structures? As part of your answer, discuss differences in terms of:
   - (i) the node data structures;
   - (ii) the operations they support;
   - (iii) any trade-offs they make in efficiency versus storage.

[6 Marks]

f) Discuss three key differences between an array and a linked list.

[3 Marks]

[END]