End of Semester 1, 2019

Curtin University

EXAMINATION PAPER CHECKLIST

for examination COMP1002 Data Structures and Algorithms

This page is to remain part of your examination file and is to be submitted with your Examination Cover Sheet and content. This page is for information only and will not be printed.

☐ Questions for the examination commence on page 3 (following the Examination Paper Checklist and Examination Cover Sheet) – Questions page should state page 1 of X.

☐ Ensure type of examination is correct 'CLOSED, OPEN OR RESTRICTED'
- CLOSED - no text books or written materials permitted
- OPEN - any text books or written materials permitted
- RESTRICTED - specified text book or written material only permitted

☐ All pages, sections and questions are numbered sequentially i.e. pages, Part A, B, C, … etc. Questions 1, 2, 3, … Subsections to question numbering is to be consistent throughout the examination paper i.e. (a), (b), (c), …

☐ General instructions to students are to be entered in the 'Instructions to Students' area of the online exam request and is reflected on the Exam Cover Sheet

☐ If there is insufficient space to enter all the general instructions to students in the 'Instructions to Students' section of the Exam Cover Sheet, the top section of page 2 may be used, preceding the commencement of the examination questions

☐ Instructions regarding the answering of questions are communicated clearly to students. e.g. Answer Part A in the answer book provided and Part B on the examination paper

☐ All questions, including subsections and parts of questions are to have marks allocated clearly. The total of all marks is to agree with the Total marks on the Exam Cover Sheet

☐ 'END OF EXAMINATION PAPER' is to be stated on the last page of the examination paper

☐ Student Name and ID is only required if the student answers on the examination paper or if the School wishes the paper to be returned

☐ Exam paper is of a high quality readable format, e.g. consistent formatting through entire document, no blurred text, images clear and printable.

The examination paper has been proof read, the above checks completed, and approved for submission.

	NAME OR ELECTRONIC SIGNATURE	DATE
EXAMINER	Valerie Maxville	15/4/19
CO-EXAMINER	Hannes Herrmann	15/4/19
HEAD OF SCHOOL/DEPARTMENT (OR DELEGATE)	Mihai Lazarescu	17/4/19

Venue _____

Student Number |__|__|__|__|__|__|__|__|

Family Name _____

First Name _____

End of Semester 1, 2019
COMP1002 Data Structures and Algorithms

Curtin University

School of Electrical Engineering, Computing and Mathematical Sciences

EXAMINATION

End of Semester 1, 2019

COMP1002 Data Structures and Algorithms

This paper is for Bentley Campus and Miri Sarawak Campus students

This is a CLOSED BOOK examination

Examination paper IS NOT to be released to student

Examination Duration 2 hours

Reading Time 10 minutes

Students may write notes in the margins of the exam paper during reading time

Total Marks 120

Supplied by the University

None

Supplied by the Student

Materials

None

Calculator

No calculators are permitted in this exam

Instructions to Students

Student to attempt all questions.

Students to write all answers on the exam paper.

For Examiner Use Only

Q	Mark
1	
2	
3	
4	
5	
6	
7	

Total _____

QUESTION ONE (Total: 12 marks): General

a) Consider a situation where you must store the name, job title and wage for all people in the 2016 Australian Census (which covers all 23 million people in the country). Individual records will need to be repeatedly retrieved later for further analysis over the course of the next several weeks of processing.

i) **(3 marks)**. What **ADT** would you use to store the data? Justify your choice.

ii) **(2 marks)**. What would you use as a **key**? Justify your choice.

b) **(3 marks)** Both Java and Python provide implementations of collections for developers to use. Give an **advantage** and a **disadvantage** of using the inbuilt collections, and an **example** of inbuilt equivalent to an ADT we have implemented in DSA. .

c) **(4 marks)**. **Polymorphism** can be used to allow simple substitution of Abstract Data Type implementations. Using an example from this unit, **explain** how you would do this, describing the class structure and the code changes required in the application.

QUESTION TWO (Total: 23 marks): Sorting and Recursion

a) **(2 marks)** Define an **in-place** sort and explain what part of the **mergesort** algorithm makes it not in-place.

b) **(2 marks)**. If you were recommended a new sorting algorithm, **monkeysort**, what would you need to know about it to decide whether to use it over existing sorts?

c) Regarding QuickSort:

i) **(2 marks)**. What is the purpose of the **pivot** element in QuickSort?

ii) **(4 marks)**. **Explain** the random and median of three pivot selection strategies. For each strategy, you must include how the strategy works, one (1) advantage and one (1) disadvantage **in comparison to the *other* strategy**.

d) **(6 marks)**. Insertion sort, mergesort and quicksort are all sorting algorithms. For each algorithm, give one (1) **advantage** and one (1) **disadvantage** in comparison to the others.

e) **(2 marks)**. Consider the following statement:

"O(N^2) sorts such as Bubble, Selection and Insertion sort should <u>never</u> be used in preference to O(N log N) sorts such as MergeSort and QuickSort".

Do you agree or disagree with this statement? **Justify** your answer.

f) Given the binarySearch code below, rewrite it to:

i) **(1 mark)**. Complete missing code (indicated by /* */)

ii) **(2 marks)**. Remove multiple returns

iii) **(2 marks)**. Provide a wrapper for the initial function call

```
private int binarySearch (int[] data,
        int toFind, int start, int end)
    {
    int mid = start + (end - start)/ 2
    if (start > end)
        return -1;
    else if (data[mid] == toFind)
        return mid;
    else if (data[mid] > toFind)
        return binarySearch(data, ***);
    else
        return binarySearch(data, ***);
}
```

```
def binarySearch (data,
                  toFind, start, end):
    mid = start + (end - start)// 2
    if (start > end):
        return -1
    elif (data[mid] == toFind):
        return mid
    elif (data[mid] > toFind):
        return binarySearch(data, ***)
    else:
        return binarySearch(data, ***)
```

QUESTION THREE (Total: 20 marks): Stacks, Queues and Lists

a) **(3 marks)** Which kind of linked list would be best suited for implementing a **queue** and why?

b) **(4 marks)** When implementing removeLast(), why isn't a **double-ended** linked list faster than a single-ended? Your answer must make reference to Big-O time complexity and space requirements in its justification.

c) **(3 marks)** Give three **benefits** of using iterators on linked lists.

d) **(10 marks)** Write an **iterator** that traverses a linked list from head to tail.

Assume the list is made up of the following ListNode objects.

```
public class ListNode
   {
     public double value = 0;
     public ListNode next = null;
   }
```

You need to write this class:

```
public class ListIterator implements Iterator
   {
```

```
class ListNode:

     def __init__(self):
        self.value = 0
        self.nextNode = None
```

QUESTION FOUR (Total: 25 marks): Trees

a) **(4 marks)**. Draw the **binary search tree** that would result from inserting the following numbers in the order that they are shown:

60, 10, 120, 50, 70, 160, 80, 130, 100, 150, 90, 140

b) **(3 marks)**. For the tree created in part (a), draw the binary tree that would result if 120 is deleted. **Describe** the steps used to perform the deletion.

c) **(4 marks)**. Using a diagram, explain the differences between a **2-3-4 tree** and a **B-tree**.

d) **(6 marks)**. Draw the **2-3-4 search tree** that results from inserting the following sequence of numbers into an initially empty tree (showing all of your working).

5, 55, 50, 500, 505, 550, 555, 5000, 55

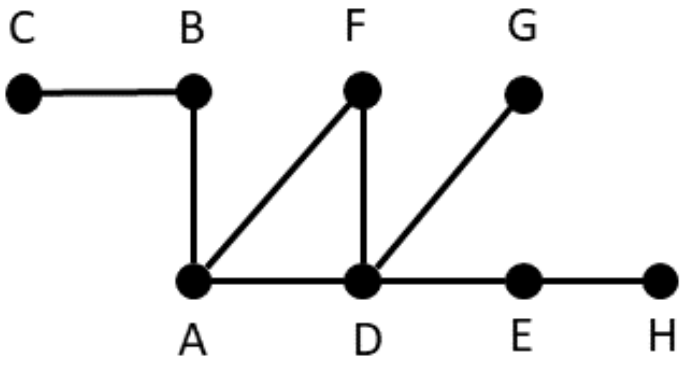
e) **(3 marks)**. Convert your final tree from 4(d) above into a **Red-Black tree**.

f) **(5 marks)**. Given the partial definition of the BinaryTree and associated TreeNode classes below, write the method **printLeaves()** to print out the values in all the leaf nodes of the tree. (Hint: you will need to traverse the tree).

```
public class BinaryTree {
    // Inner class TreeNode
    private class TreeNode {
        public int value;
        public TreeNode left;
        public TreeNode right;
    }

    // Class BinaryTree
    private TreeNode root;

    public void printLeaves(){
        // You must implement this
    }

    // C'tors and other methods are not relevant to the question
}
```

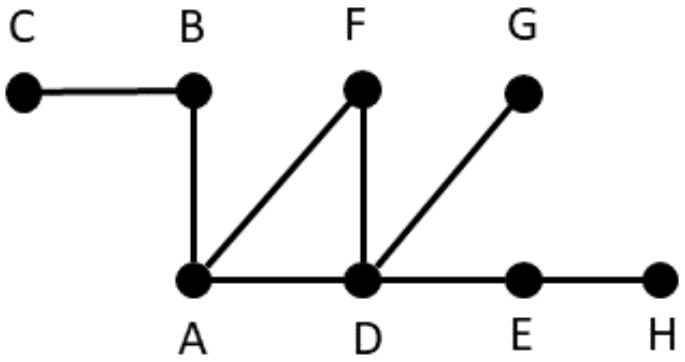
```
class BinaryTree():
    def __init__(self):
        self.root = None

class TreeNode():
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

def printLeaves():
    # You must implement this
```

QUESTION FIVE (Total: 15 marks): Graphs



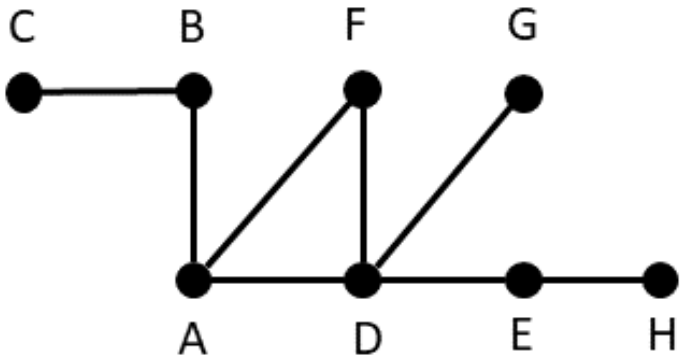
a) **(3 marks)**. Provide the **adjacency matrix** representation for the above graph.

b) **(3 marks)**. Provide the **adjacency list** for each vertex of the above graph.



(duplicate of graph from previous page)

c) **(3 marks)**. Perform a **depth-first** traversal of the graph above, starting with vertex A. Select edges in alphabetical order. List the **edges** in the order they are visited.



(duplicate of graph from previous page)

d) **(3 marks)**. Perform a **breadth-first** traversal of the graph above, starting with vertex A. Select edges in alphabetical order. List the **edges** in the order they are visited.

e) **(3 marks)**. Graphs can be implemented in many ways. Describe and justify a situation when you would create **edge** classes and objects in your graph.

QUESTION SIX (Total: 15 marks): Heaps

a) Given the following list of numbers:

10, 40, 15, 5, 25, 20, 45, 10, 90

i) **(5 marks)**. Draw the **heap** (as a tree diagram) that would be built if the above numbers were inserted into a **max heap** in the order they are listed. Show your working.

ii) **(2 marks)**. Convert the heap tree in your answer from part i) into an **array** representation of the tree.

iii) **(2 marks)**. Show what the **array** form of the heap would look like after inserting the value of **3** into the heap (hint: use your tree to help you trace the 'trickle-up').

b) **(3 marks)**. Give three (3) differences between a **heap** and a **binary search tree**.

c) **(3 marks)**. Describe in detail the steps involved in *removing* an item from a **Max heap**. Note that you must describe how each step works in enough detail to gain full marks. A diagram may help.

QUESTION SEVEN (Total: 10 marks): Hashing

a) **(4 marks)**. Given the following hash function that maps an integer key to an index given a table of length 11:

```
public int hash(int key) {
  int hashIdx;
  hashIdx = key % 11;
  return hasIdx;
}
```

```
def hash(key):
  hashIdx = key % 11
  return hashIdx
```

Use the above hash function to insert the following key **:** value pairs into the hash table below (copy the table to your exam booklet). Handle collisions using the **quadratic probing** method.

15 : “First” 5 : “Second” 26 : “Third” 13 : “Fourth” 4 : “Fifth”

Index	*Key*	*Value*
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

b) **(4 marks)**. Explain why high load factors have a different impact when using **separate chaining** as compared to **open addressing**.

c) **(2 marks).** What **four properties** do we look for in a good **hashing** function?

END OF EXAMINATION