Venue	____________________
Student Number	|__|__|__|__|__|__|__|__|
Family Name	____________________
First Name	____________________

Curtin University

School of Electrical Engineering, Computing and Mathematical Sciences

EXAMINATION

End of Semester 2, 2018

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

Answer all questions

For Examiner Use Only

Q	Mark
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

Total ________

QUESTION ONE (Total: 24 marks): Sorting and Heaps

a) **(6 marks)**. A priority queue can be implemented using one of the following three data structures:

- Array
- Linked list
- Heap

i) For each of the above data structures, give one (1) advantage and one (1) disadvantage of using it as the basis for a priority queue.

ii) **(2 marks)**. If you had to write a priority queue, which of the three data structures from part (i) would you choose to implement the priority queue? Explain your reasoning.

b) **(3 marks)**. HeapSort and MergeSort are both O(N log N) sorting algorithms in all cases (best, average and worst cases). If you had to implement a sorting algorithm using one of these two, which algorithm would you choose? Why?

c) **(4 marks)**. A heap ADT is a type of binary tree, but one which can be implemented using:

- An actual binary tree data structure, OR
- An array organised such that it represents the tree.

Explain why it is more common to implement heaps using the array form.

d) The following code (Java and Python) implement SelectionSort, but are **wrong**.

```
void selectionSort(int[] A) {
   int minIdx;
   for (int nn = 0; nn < A.length-1; nn++) {
      minIdx = nn;
      for (int ii = nn+1; ii < A.length; ii++) {
         if (A[ii] < A[nn])
            minIdx = ii;
         }
      A[nn] = A[minIdx];
      A[minIdx] = A[nn];
      }
   }
```

```
def selectionSort(A):
   for nn in range(0, len(A-1)):
     minidx = nn
     for ii in range(nn+1,len(A)):
        if A[ii] < A[nn]:
           minidx = ii

     A[nn] = A[minidx]
     A[minidx] = A[nn]
```

i) **(5 marks)**. What would be the result of trying to sort the following set of numbers with the above flawed code? (e.g. input values [6,9,2,7,4,8,2])

ii) **(4 marks)**. Rewrite the SelectionSort function to work correctly.

END OF QUESTION ONE

QUESTION TWO (Total: 16 marks): Recursion and Decisions

a) **(6 marks)**. The following code calculates the value of the nth element in the Fibonacci sequence. Don't worry about wrappers and validation - this time.

```
public double fib(int n) {
     double fibVal;
     if (n == 0)
        fibVal = 0
     else if (n == 1)
        fibVal = 1;
     else
        fibVal = fib(n-1) + fib(n-2)
     return fibVal;
     }
```

```
def fib(n):
    if (n == 0):
        fibVal = 1
    elif (n == 1):
        fibVal = 1
    else:
        fibVal = fib(n-1) + fib(n-2)
    return fibVal
```

The above functions (Java and Python) are inefficient in that they re-calculate lower values of fib() multiple times in different recursive sub-branches. One way to solve this issue whilst still retaining a recursive algorithm is to also pass (as a parameter) an array of size N to each recursion, where this array is used to cache (store) the already-calculated Fibonacci values (use a value of –1 to indicate that it hasn't been calculated yet). Rewrite the function (in Java or Python) to incorporate such caching.

Hint: Use one of the following function headers:

`double fib(int n, int[] fibCache)` **OR** `fib(a, fibCache)`

You may assume that `fibCache` is of size n and initialised to all –1's on the first call

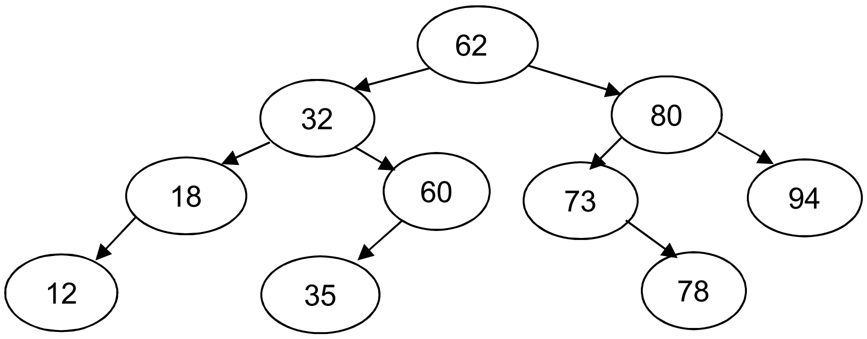
b) **(6 marks)** When saving data to a file, we can use a variety of formats. Describe the differences between serialization and comma separated text files (csv) and give two advantages and two disadvantages of each approach.

c) **(4 marks)** Both Java and Python provide implementations of collections, such as queues for developers to use. Give an advantage and a disadvantage of using the inbuilt collections. Discuss an example of when you might choose to implement your own queue.

END OF QUESTION TWO

QUESTION THREE (Total: 31 marks): Trees

a) For the tree drawn below, draw the binary tree that would result if the following operations were applied to it (apply each operation in succession to build up the tree).



i) **(4 marks)**. Insert 68, 48, 38, 64

ii) **(3 marks)**. Delete 60

b) **(3 marks)**. Give the sequence of numbers that (when inserted in order) will create the original tree shown in part (b).

c) Concerning complete trees:
   i) **(3 marks)**. Explain why it is desirable to have a completely-balanced tree.

ii) **(3 marks)**. Explain why an almost-complete tree is just as desirable as a completely-balanced tree.

d) **(5 marks)**. A tree can also be represented using an array data structure – this is often the case for heaps, but can apply to any binary tree. Define the array that would represent the original tree from part (b). (Hint: Use –1 to indicate null nodes).

d) **(5 marks)**. Given the partial definition of the BinaryTree and associated TreeNode classes below, write the method **total_odd()** to add up all of the odd values in 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 int total_odd(){
      // 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 total_odd():
    # You must implement this
```

e) **(5 marks)**. Given the previous partial definition of the BinaryTree and associated TreeNode classes, write the method **traverseReverse()** that will print the values in the tree in **reverse** order.

.

END OF QUESTION THREE

QUESTION FOUR (total: 24 marks): Hash Tables

a) **(4 marks)**. What is the purpose of the hash function in a hash table? What makes a “good” hash function?

b) Concerning collisions:
   i) **(2 marks)**. What is a collision in a hash table?

ii) **(8 marks)**. List four types of collision handling in a hash table, explain how each one works and give one (1) advantage that each has in comparison to one (or more) of the other types.

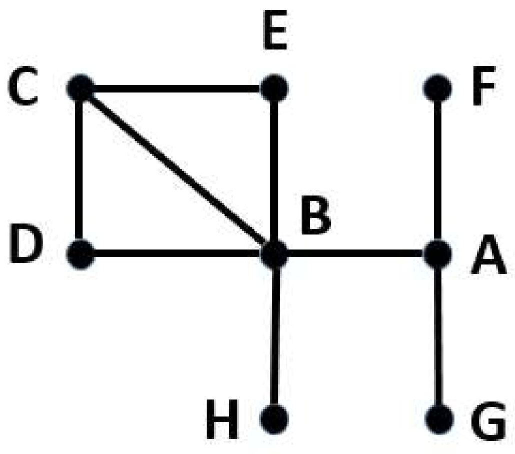
c) **(4 marks)**. Given that a hash table is O(1) access time and a binary tree is O(logN) access time, would you ever use a binary tree instead of a hash table? Explain.

d) Concerning load factor
   i) **(2 marks)**. What is a hash table's *load factor*?

   ii) **(4 marks)**. What is the effect of a load factor that is approaching 1.0 on the speed of inserting and accessing elements in the hash table? Why?

END OF QUESTION FOUR

QUESTION FIVE (total: 12 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.

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.

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.

END OF QUESTION FIVE

QUESTION SIX (total: 13 marks): Advanced Trees

a) **(6 marks)**. Draw the 2-3-4 search tree that results from inserting the following sequence of numbers into an initially empty tree (i.e. one tree per insertion, showing all of your working).

15, 60, 45, 50, 20, 100, 75, 20

b) **(3 marks)**. Convert your final 2-3-4 tree from (b) above into a Red-Black tree.

c) **(4 marks)**. A B-Tree can have any number of data items per node. Discuss in detail how the number of data items is chosen for a B-Tree. You may use diagrams to assist your explanation.

END OF EXAMINATION