JFK
(110)

THE UNIVERSITY OF THE WEST INDIES
ST. AUGUSTINE

EXAMINATIONS OF December 2018

Code and Name of Course: COMP 2611 – Data Structures

Date and Time: Monday 17th December 2018 1 pm Duration: 2 Hours

INSTRUCTIONS TO CANDIDATES: This paper has 5 pages and 4 questions

Answer ALL Questions

1) Assume that a node in a binary tree is defined as follows:

```
struct BTNode {
      int data;
      BTNode * left;
      BTNode * right;
      BTNode * parent;
};
```

a) A binary tree is said to be a *full binary tree* if every non-leaf node has exactly two non-empty subtrees. Write a function with the following prototype which returns *true* if the binary tree rooted at *bt* is a full binary tree, and *false* otherwise.

```
bool isFullBT (BTNode * bt);
```

[5 marks]

b) Suppose node *X* has to be deleted from a binary search tree (BST). One of your colleagues has suggested that if *X* has two non-empty subtrees, the deletion can be done by replacing *X* with its in-order predecessor **instead of** its in-order successor, provided that equivalent rules for re-connecting sub-trees of the in-order predecessor are followed.

Using this approach, draw separate diagrams to show the BST in Figure 1 after each of the following nodes has been deleted (assume that the deletions are independent of each other): 16, 15, 5. [4 marks]

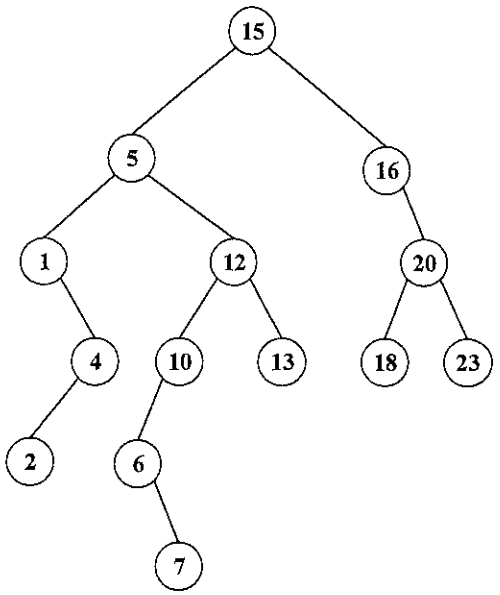


Figure 1

Question 1) continues on the next page.

c) Write a function with the following prototype which accepts an array of integers as a parameter and sorts the array of integers using a BST. You **cannot** use a heap or a typical sort algorithm. Stack, Queue, and BST functions are available for use via function calls, in case you need to use them.

```
void sort (int A[], int lengthA);
                // lengthA is the number of elements in A
```

[6 marks]

Total marks 15

2) a) In the quicksort algorithm, the partition function accepts three parameters: the array to be sorted A, and two locations in the array, m and n, where $m < n$. The partition function works by choosing a pivot value p, between m and n, such that all elements to the left of p are less than or equal to p and all elements to the right of p are greater than p.

Assuming that the pivot is chosen to be $A[n]$, illustrate the operation of the partition function on the following array A, where m is 0 and n is 7.

2	8	7	1	3	5	6	4
0	1	2	3	4	5	6	7

[5 marks]

b) Consider the *minHeapify* function below that operates on an array of integers:

```
void minHeapify (int A[], int heapSize, int i);
```

Assuming that the left and right subtrees of node i are min-heaps, the function maintains the min-heap property starting at node i. On completion, the elements of the array A from 1 to *heapSize* comprise a min-heap.

i) Write the code for the *minHeapify* function. [5 marks]

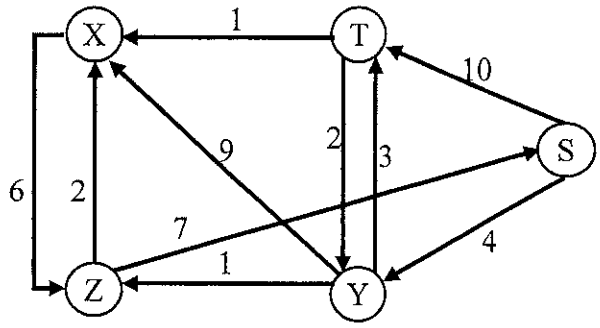
ii) Suppose that the array A initially contains the values shown below:

	2	16	5	9	4	8	11	15	19	7
0	1	2	3	4	5	6	7	8	9	10

Illustrate the action of the *minHeapify* function on A by drawing the binary tree corresponding to A, **each** time it is modified by the *minHeapify* function. [2 marks]

Total Marks 12

3) a) Given the following directed graph:



i) Draw the adjacency list representation of the graph. List the nodes in alphabetical order. [2 marks]

ii) Derive the minimum-cost paths from vertex **S** to every other vertex using *Dijkstra's* algorithm. For each vertex, give the cost and the path to get to the vertex. At each stage of the derivation, show the minimum cost fields, the predecessor, the element at the top of the priority queue, and the other elements in the priority queue. You must use the sheet provided in answering this question. [10 marks]

b) The following are some declarations for a directed graph represented as an adjacency list:

```
struct Edge {
   int dest;
   int weight;
   Edge * nextEdge;
};

struct Vertex {
   string ID;
   int colour;
   Edge * firstEdge;
};

struct Graph {
   int numVertices;
   Vertex vertices[100];
};
```

i) Write a function, *hasEdge*, with the following prototype which returns *true* if there is an edge between vertex *u* and vertex *v* in the graph *g*, and *false* otherwise:

```
bool hasEdge (Graph * g, int u, int v);
```

[3 marks]

ii) The *in-degree* of a vertex *v* in a graph *g* is the amount of edges entering *v*. Using the *hasEdge* function from b) i) or otherwise, write a function, *inDegree*, with the following prototype which returns the in-degree of a vertex *v*:

```
int inDegree (Graph * g, int v);
```

[3 marks]

Total marks 18

4) A certain *undirected* graph *G* has *n* vertices. An *n* x *n* adjacency matrix **A** can be used to store the weight of the edges in *G* (assumed positive). However, it is more efficient to use a one-dimensional array.

a) Explain how the data in **A** can be stored in a one-dimensional array **B**, conserving storage as much as possible. [3 marks]

b) Write a function, *edgeWeight*, which accesses **B** to return the weight of the edge between vertex *i* and vertex *j*. If there is no such edge, it returns -1. The prototype of *edgeWeight* is as follows:

```
int edgeWeight (int i, int j);
```

[4 marks]

c) Using the *edgeWeight* function from b), write another function *minCostVertex*, which, given a vertex *u*, finds the edge with the least cost weight from *u* and returns the corresponding vertex. If there is no such edge, it returns -1. The prototype of *minCostVertex* is as follows:

```
int minCostVertex (int u);
```

[3 marks]

Total marks 10

End of Examination

(Total Marks = 55)