EASTERN UNIVERSITY, SRI LANKA
DEPARTMENT OF MATHEMATICS
SECOND EXAMINATION IN SCIENCE – 2016/2017
FIRST SEMESTER (October/November, 2018)
CS 201- Data Structure and Algorithm Design

Answer all questions **Time Allowed: 2 Hours**

Q1. Analysis of algorithms can be stated as determining the amount of resources (such as time and storage) necessary to execute them.

a) Define the term *data structure*. [8 marks]

b) *Distinguish* the following three terms in plain English:

i. *Algorithm*;

ii. *Pseudocode;*

iii. *programme*. [18 marks]

c) Arithmetic expression can be written in *infix*, *prefix* and *postfix* notations. Write down the following expressions in *prefix* and *postfix* notations.

i. $A + B * C + D$;

ii. $A * (B + (C / D))$;

iii. $((A + B) * C) - D$. [24 marks]

d) Consider sorting *n* numbers stored in an array *A* by first finding the smallest element of *A* and exchanging it with the element in *A[1]*. Then find the second smallest element of *A*, and exchange it with *A[2]*. Continue in this manner for the first *n−1* elements of *A*.

i. Write a *pseudocode* for this algorithm. [16 marks]

ii. Write down the *loop invariant* that this algorithm maintains. [10 marks]

iii. Explain why the algorithm needs to run for only the first $n-1$ ele

rather than for all n elements. [12 r

iv. Give the *best-case* and the *worst-case* running times of the

algorithm in *theta notation.* [12 r

Q2. A *tree* is a widely used abstract data type that simulates a hierarchical tree stru with a root value and subtrees of children with a parent node, represented as a linked nodes.

a) Write definitions of the following types of binary trees:

i. Strictly binary tree;

ii. Complete binary tree;

iii. Extended binary tree. [15 m

b) Briefly describe the following types of tree traversals with the aid of exampl

i. Pre-order traversal;

ii. Post-order traversal;

iii. In-order traversal. [18 m

c) Let BT is a binary tree of 13 nodes:

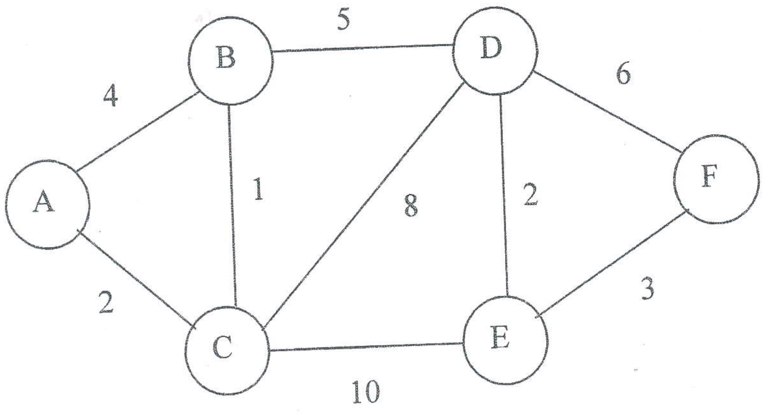
The *post-order traverse* visits the nodes in the order: D F E B G L K L H

and the *in-order traverse* visits the nodes in the order: D B F E A G C L J

Show the *binary tree* that satisfies the above, and write the nodes in the ord

which the *pre-order traverse* visits. [15 m

d) Consider the graph given below:

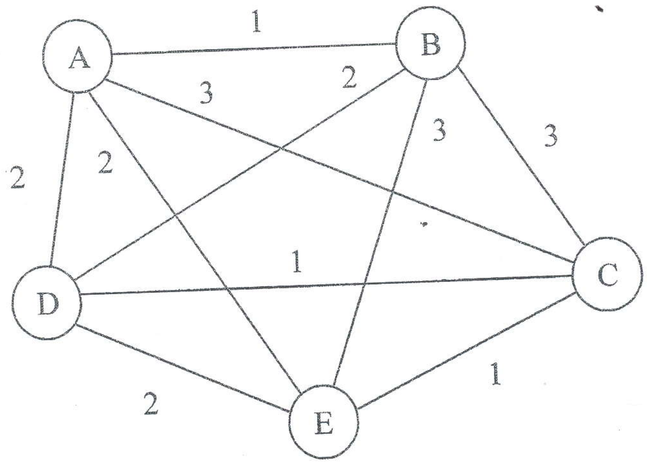


i. Show how this graph would be represented as an *adjacency matrix.*

[10 marks]

ii. Draw the *shortest-path tree* produced by running the *Dijkstras algorithm* on this graph, using vertex *A* as the source. [15 marks]

e) Give an algorithm to identify a *minimum cost spanning tree* in a connected graph *G(V, E)*. Applying your algorithm, find a minimum cost spanning tree from the following graph:



[27 marks]

Q3. Sorting refers to ordering data in an increasing or decreasing fashion according to some linear relationship among the data items.

a) Write an *iterative algorithm* for merging two sorted lists of numbers to form a single sorted list. [20 marks]

b) Write a *recursive algorithm* for *merge sort.* [20 marks]

c) Show the steps of *Quick sort* on the following list of unsorted integers. Assume that the *pivot* node is always the one on the left-hand side of the list. Clearly indicate your choices of *pivots* and underline all numbers that have been placed in their correct respective positions.

87, 36, 22, 15, 56, 85, 48, 90, 72, 6 [20 marks]

d) Both *quick sort* and *merge sort* have an expected time of *O(n log n)* to sort a list containing *n* items. However, in the worst-case, quick sort can require $O(n^2)$ time, while merge sort's worst-case running time is the same as its expected time, *O(n log n)*.

i. What accounts for the difference in worst-case times? How is it that quick sort can require $O(n^2)$ time, while merge sort always guarantees *O(n log n)*? [20 marks]

ii. Given that merge sort's worst-case time is better than quick sort's, why is quicksort so commonly used in practice? [20 marks]

Q4. A *stack* is a container of objects that are inserted and removed according to the *last-in first-out (LIFO)* principle.

a) Consider the following sequence of stack operations:

push(d), push(h), pop(), push(f), push(s), pop(), pop(), push(m).

i. Assume the *stack* is initially empty, what is the sequence of popped values, and what is the final state of the *stack*? (Identify which end is the top of the *stack.*) [15 marks]

ii. Suppose you were to replace the *push* and *pop* operations with *enqueue* and *dequeue* respectively. What would be the sequence of dequeued values, and what would be the final state of the queue? (Identify which end is the front of the queue.) [15 marks]

b) Use a *stack* to test for balanced parentheses, when scanning the following expressions. Your solution should show the state of the stack each time it is modified. The "state of the stack" must indicate which is the top element.

Only consider the parentheses [,], (,), {,}. Ignore the operands and operators.

i. [a + { b / (c - d) + e / (f + g) } - h]

ii. [a { b + [c (d + e) - f] + g } [24 marks]

c) Suppose you have three stacks *s1*, *s2*, *s2* with starting configuration shown on the *left*, and finishing condition shown on the *right*. Give a sequence of *push* and *pop* operations that take you from start to finish. For example, to pop the top element of *s1* and push it onto *s3*, you would write *s3.push(s1.pop())*.

Start			**Finish**		
A					B
B					D
C					A
D					C
………	………	………	………	………	………
s1	s2	s3	s1	s2	s3

[20 marks]

d) Assume you have a stack with operations: *push( )*, *pop( )*, *isEmpty( )*. How would you use these stack operations to simulate a *queue*, in particular, the operations *enqueue( )* and *dequeue( )*?

Hint: Use two stacks, one of which is the main stack and another one is a temporary stack. [26 marks]