# UNIVERSITY OF MORATUWA

## FACULTY OF ENGINEERING

## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

BSc Engineering Honours Degree
2012 Intake Semester 2 Examination

# CS 2022: DATA STRUCTURES AND ALGORITHMS

Time allowed: 2 Hours June 2014

**ADDITIONAL MATERIAL:** *None*

**INSTRUCTIONS TO CANDIDATES:**

1. This paper consists of **FOUR (4)** questions in **FIVE (6)** pages.
2. Answer **ALL** QUESTIONS
3. Start answering each main question on a new page.
4. All questions carry equal marks.
5. This examination accounts for 60% of the module assessment.
6. This is a closed book examination.

   ***NB: It is an offence to be in possession of unauthorised material during the examination.***
7. Only calculators approved and labelled by the Faculty of Engineering are permitted.
8. Assume reasonable values for any data not given in or with the examination paper. Clearly state such assumptions made on the script.
9. In case of any doubt as to the interpretation of the wording of a question, make suitable assumptions and clearly state them on the script.
10. This paper should be answered only in English.

*Continued…*

**Q1. [25 *marks*]**

a)

i. What is an algorithm? [2 *marks*]

ii. List four (4) factors that should be considered in evaluating an algorithm. [2 *marks*]

b) Give the asymptotic growth in "big oh" notation for the following functions. ***Please show how you obtained your answer***.

i. $T(n) = 1222\,n + (3\times10\text{-}22)\,n^2 + 0.9n \log n$ [2 *marks*]

ii. $T(n) = 3T\left(\frac{n}{2}\right) + n \log n.$ [3 *marks*]

iii. $T(n) = T\left(\lfloor\sqrt{n}\rfloor\right) + n$ [4 *marks*]

c) Show how the following array will be sorted using the Merge sort.

[ 24, 56, 64, 78, 34, 22, 45, 31, 96 ] [5 *marks*]

d) Analyze the worst case time complexity of the optimized version of bubble sort is shown below. [7 *marks*]

```
OPTIMIZED-BUBBLE-SORT(A)
   for j = A.length to 2
        swapped = false
        for i = 2 to j
             if A[i-1] > A[i]
                  temp = A[i]
                  A[i] = A[i-1]
                  A[i-1] = temp
                  swapped = true
        if (!swapped)
                  break;
```

**Q2. [25 *marks*]**

a) What is an abstract data type (ADT)? [4 *marks*]

b) Give one advantage and one disadvantage of using a linked list structure to implement a queue rather than an array. [2 *marks*]

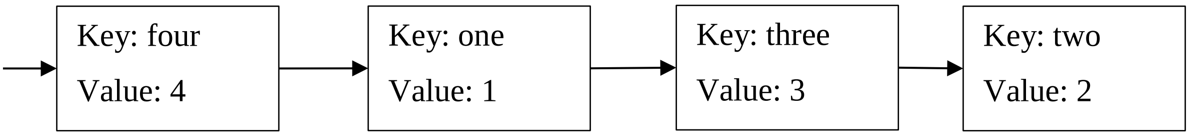
c) You are provided with an implementation of a (single) linked list which supports the following operations.

- LIST-SEARCH($L$,$k$): Finds the first node with the key $k$ in the list $L$. Returns NIL if no element is found.
- LIST-HEAD($L$): Returns the value of the first node in the list.

*Continued…*

- LIST-INSERT(*L*,*x*): Inserts the value *x* as the first element of the list *L*.
- LIST-DELETE(*L*,*x*): Deletes the first node which contains value *x* from the list *L* if it exists.
- LIST-INSERTAT(*L*,*x*,*i*): Inserts the value *x* into the $i^{\text{th}}$ location of the list *L*.
- LIST-DELETEAT(*L*,*i*): Deletes the node in the $i^{\text{th}}$ location of the list *L*.
- LIST-GET(*L*,*i*): Returns the value of the node in the $i^{\text{th}}$ location of the list *L*.

You are required to implement a general dictionary structure similar to the dictionary data structure in Python programming language. The dictionary data structure allows you to store a list of key-value pairs. The key will be a string and the value can be of any data type. In the dictionary, the ***data items are stored in the order of the keys***. A sample dictionary which stores key value pairs { <"one", 1>, <"two", 2>, <"three", 3>, <"four", 4> } is shown below.



Using the provided linked list implementation as a library (you are not allowed to change any operations of the provided linked list implementation), write the pseudo code to implement the dictionary data structure with following operations.

iv. Initialize(*D*): Initialize the dictionary to an empty dictionary. [3 *marks*]

v. Insert(*D*, *k*, *v*): Should insert the key-value pair <*k*,*v*> to the proper location in the dictionary *D*. [4 *marks*]

vi. Delete(*D*, *k*): Should delete the element which has the key *k*, if it exists in the dictionary. [3 *marks*]

vii. Get(*D*, *k*): Should return the value of element which has the key *k*, if it exists in the dictionary. [3 *marks*]

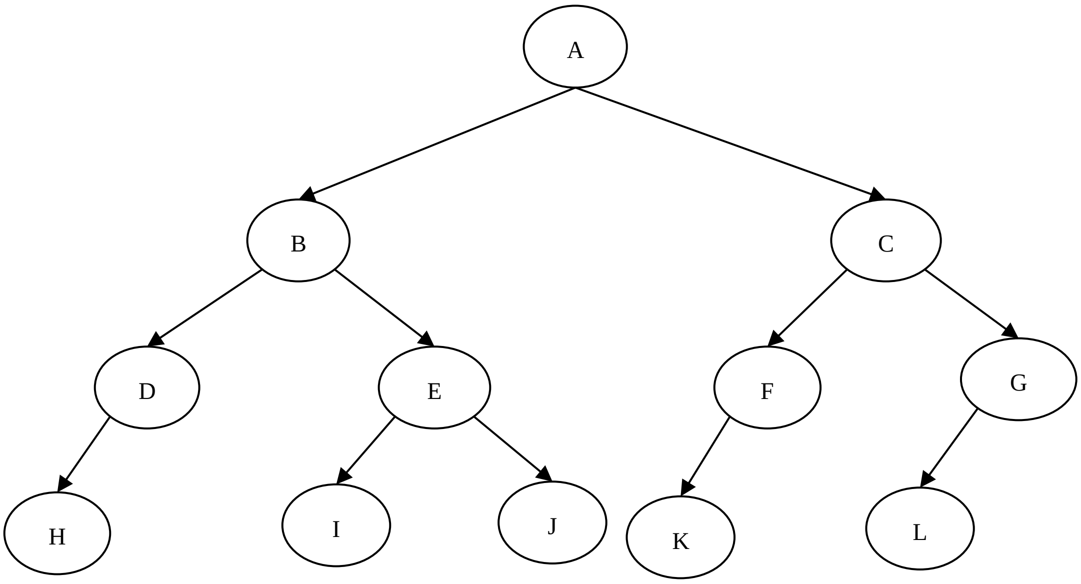
viii. Size(*D*): Should return the number of key-value pairs in the dictionary. [2 *marks*]

d) For the binary tree shown below, write the order the nodes will be processed if the tree is traversed;

i. In- order [2 *marks*]

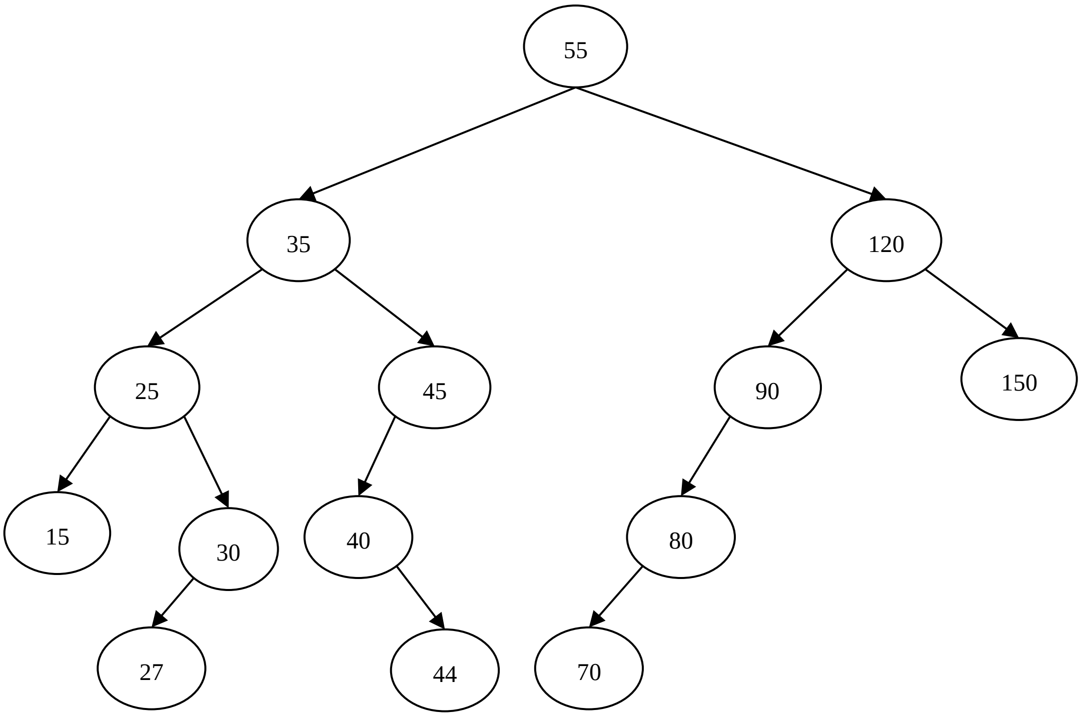
ii. Post-order [2 *marks*]

*Continued…*



**Q3. [25 *marks*]**

b) Consider the binary search tree shown below.



Show how the following values can be deleted from the tree. Clearly explain the deletion process.

i. 90 [2 *marks*]

ii. 35 [3 *marks*]

*Continued…*

c) Trees can be implemented in different ways. Briefly explain three (3) ways of implementing K-ary trees. You should explain **how the tree structure is maintained** and the **advantages and limitations** of each approach. [6 *marks*]

d) Write pseudo code to implement a min-heap. Your heap implementation should support the following functions.

i. Min-Heapify(*A*, *i*): Given a node *i*, with left child *l* and a right child *r* and with both the sub -trees rooted by *l* and *r* are max-heaps, this function should convert the sub-tree rooted by node *i* to a min-heap. [3 *marks*]

ii. BuildMin-Heap(*A*): This function should convert the specified array into a min-heap. [3 *marks*]

iii. HeapExtractMin(*A*): This function will remove the maximum element from the heap and return it. [2 *marks*]

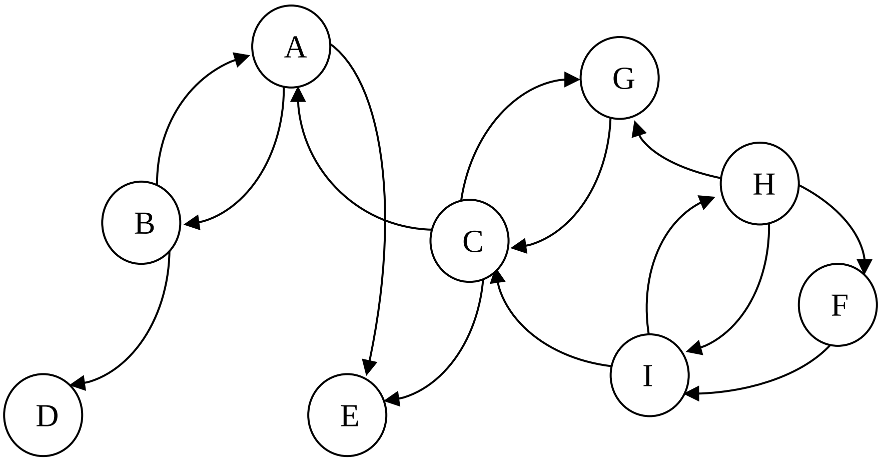
iv. Min-HeapInsert(*A*, *ne* ): This function should insert the specified element to the min heap. [3 *marks*]

v. HeapIncreaseKey(*A*, *i*, *amt*): This function will increase the key value of the node *i* in the heap by the amount specified by *amt*. [2 *marks*]

vi. HeapDecreaseKey(*A*, *i*, *amt*): This function will decrease the key value of the node *i* in the heap by the amount specified by *amt*. [1 *mark*]

**Q4. [25 *marks*]**

a) Perform the Depth First Search in the following graph. [6 *marks*]



.

*Continued…*

Bellman-Ford algorithm and Dijkstra's Algorithm are two algorithms for calculating Single Source Shortest Path (SSSP).

b) Explain how a SSSP algorithm can be used to solve the single destination shortest path problem in which you are expected to calculate the shortest path to a specified vertex starting from each of other vertices. [4 *marks*]

c) Using one of the above algorithms, calculate the shortest path to **F** from all other vertices in the graph shown in following diagram (Figure Q4). Clearly indicate all the steps you followed. [10 *marks*]

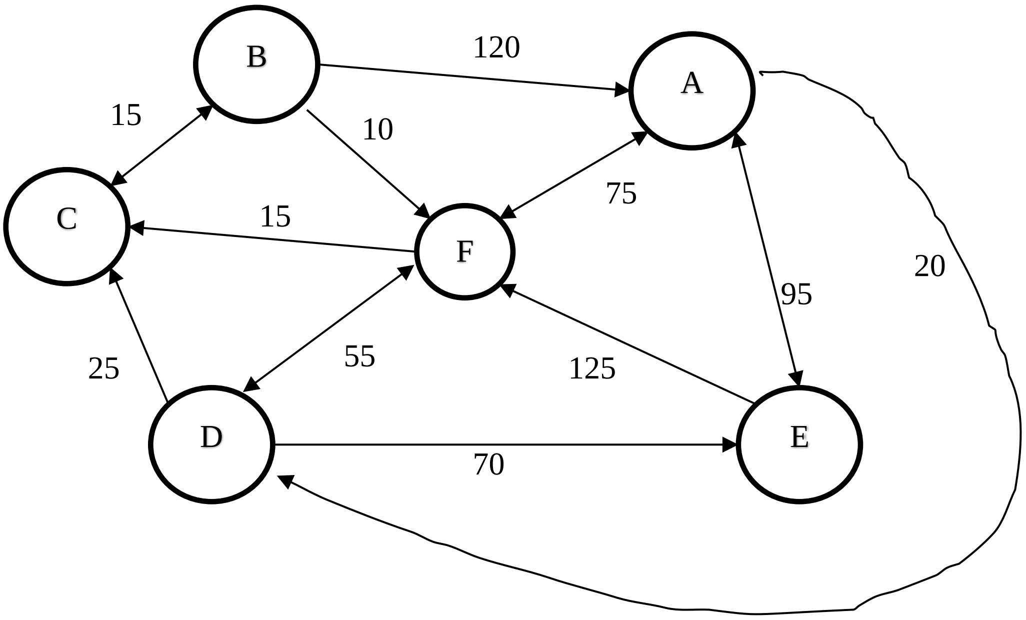


**Figure Q4**

d)

i. What is the greedy choice property? [2 *marks*]

ii. Do greedy solutions always outperform dynamic programming solutions in terms of time required to calculate the solution. Explain your answer. [3 *marks*]

--- ***End of the Paper*** ---