Assignment Brief

Data Structures and Algorithms

COS5021-B

Coursework - (70%)

School of Computing and Engineering
University of Bradford

Module Title: Data Structures and Algorithms

Module Code: COS5021-B
Module Credit: 20
Level: L5
Academic Year: 2025/26

Date due: 12 Dec 2025, 15:00
Length limit: There is no limit whereas each question should be explained for each steps performed

Learning Outcomes:
This assessment will evaluate follow learning outcomes of the module:

LO3: Identify algorithms and demonstrate their operation and effect in solving problems.

LO4: Assess efficiency of various implementations and justify choice of implementation for any application.

LO5: Construct and test simple algorithms using suitable data structures, analyse the complexity and running times for algorithms

Assessment Methods:
Coursework - This consists of **six (06)** tasks, and you need to answer all the questions according to the question asked. **Note:** Understanding the questions is part of the assessment and no AI tools would be used for answering this assessment.

Please take note of the following regulations/advice where appropriate:

- Check the assignment marking criteria when doing your assessment.
- Go through your lecture/lab materials on the canvas for more understanding.

Guidelines: Prepare a report for each solution based on the guidelines explained below. Include a cover sheet with your name and UoB number. The coursework must be submitted using Canvas by the deadline. Please name your file with your name and UoB number. If you have problems with Canvas, please contact the Module Coordinator as soon as possible.

Task 01: Stack–Queue Algorithm Trace (Total Marks: 20)

Consider the following pseudocode:

```
while not Stack.isEmpty():
    x ← Stack.pop()
    if x < 20 then
        Queue.enqueue(x * 2)
    else
        Stack.push(x - 5)
```

Assume the initial contents of the two data structures are as follows:

- **Stack:** [10, 25, 15, 30] (top = 30)
- **Queue:** [] (empty)

Trace the execution of the above pseudocode step-by-step, starting from the given initial state. For each step, you must clearly **show** and **explain** the operation being performed (pop, push, enqueue, etc.). Indicate the value of x retrieved or modified at that step.

Draw or represent the state of the Stack (top → bottom) and Queue (front → rear) after the operation. Continue tracing until the stack becomes empty. Beside the step-by-step explanation, you should write a short explanation at the end of what is happening and why that change occurs. **Note**: Without your explanation the answer would be not fully reward with the marks.

Task 02: Binary Search Tree — Deletion and Structural Reasoning (Total Marks: 15)

You are required to build, modify, and analyse a Binary Search Tree (BST) using the following dataset:

Key: [40, 25, 60, 10, 30, 50, 70, 55, 65]

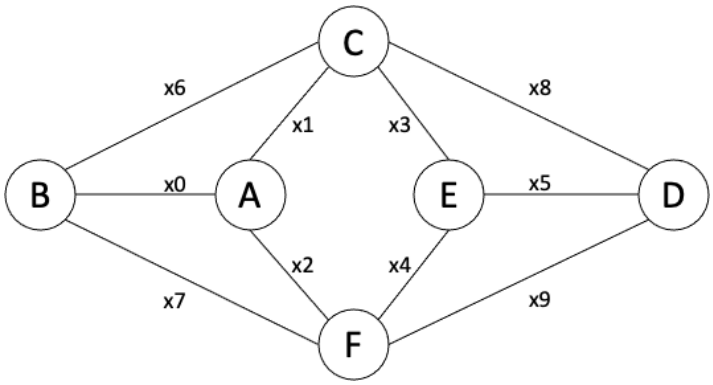
Construct and annotate the BST while inserting the elements in the given order into an initially empty BST. Draw the final BST clearly, showing all node keys. Label the following for each node: (Height - distance from leaf, whether it has 0, 1, or 2

children). Briefly explain how insertion maintains the BST property at each step. Moreover, you need to perform and write out the in-order and pre-order traversals of your tree with each step explained in your own words. Further, you need to perform a **deletion of the node with key = 60,** and for each step identify the case that applies such as (**Case 1:** Leaf node deletion, **Case 2:** Node with one child, **Case 3:** Node with two children). Finally, you need to Redraw the BST after deleting node 60, showing which nodes were affected. **Note:** All steps need to be explained clearly and properly, failing this would result in reduction of the marks.

Task 03: Dijkstra's algorithm (Total Marks: 15)

Based on the key-based weighted graph below, please apply Dijkstra's algorithm to search the shortest path starting from node A.

Key = [74,7,84,90,94,81,99,48,17,36]



Show each step by table and, at each step, the calculation and the two sets of nodes and distances; one set for which shortest distances have been determined and the other set of the remaining nodes and distances. Moreover, analyse the complexity of Dijkstra's algorithm with proper justification. **Note:** Each step should be backup with clear and proper explanation, failing this will reduce the marks.

Task 04: Hash Table (Total Marks: 15)

The task here is to show a trace of the operations needed to insert objects with the key {key : [22, 12, 13, 24, 14, 16, 27, 20, 31, 10]}, one by one, into an initially empty 11-bucket hash table with (primary) hash function h1(x) = x mod 11. Any collision will be resolve through using the secondary hash function h2(x) = (x mod 3) + 1.

Your submission should have the section heading 'Hash Table trace 1' followed by the coded trace of operations:

- Pn to probe bucket n (to see if it already contains an entry);
- Ixx@n to insert key xx into bucket n;

with the coded operations in sequence on successive lines, e.g.

Hash Table trace
P3
I25@3
P10
 I10@10
 P3
P4
 I14@4
...
Moreover, you need to construct pseudocode/algorithm for the entire process showing how to insert the values in the bucket and how you deal with all the collisions. **Note:** Each step should be explained clearly and properly especially the point of collision.

Task 05: AVL Tree (Total Marks: 20)

Insert the following sequence of keys one by one into an initially empty AVL tree:

Keys: [53, 65, 40, 79, 69, 92, 50, 30, 10, 55]

After each insertion, perform any necessary **rotations** to restore the AVL balance property.

Your submission should have the section heading 'AVL trace' followed by the coded trace of operations:

- Ixx to insert key xx at the root of the previously empty AVL tree;
- IxxLyy to insert key xx as the left child of the node containing key yy;
- IxxRyy to insert key xx as the right child of the node containing key yy;
- Rxx to rotate the node containing key xx with that of its parent (in order to restore AVL balance);

with coded operations in sequence on successive lines, e.g.

```
AVL
trace
I25
I10L25
I15R10
R15

R15

...
```

Each code should appear on a new line, and your trace should show all insertions and rotations in exact chronological order. For double rotations (LR or RL), show both rotations separately (e.g., R60 followed by R65). For **each operation in your trace**, you must also: Indicate on each node its:

- **Height** (leaf = 0)
- **Balance factor** = height(left) − height(right)

For each rotation, draw an additional before and after subtree diagram, and write: - The type of imbalance (LL, RR, LR, RL) and proper justification. Finally, you should provide a diagram of the final AVL tree (after all insertions), its in-order traversal (should produce the sorted key order) and the height of the final tree.

Note: you should provide clear and properly explanation for each rotation, failing this will reduce the marks.

Task 06: 2–4 Tree Construction and Deletion (15 marks)

You are required to construct and manipulate a 2–4 tree (multiway search tree where each node can have 2, 3, or 4 children and contain 1–3 keys). Each insertion and deletion must maintain 2–4 tree balance properties. What you need to do is to create an empty 2-4 Tree and insert the keys sequentially.

Key: [35, 60, 25, 40, 70, 20, 10, 90, 80, 30, 45, 50, 55]

Once all the keys have been inserted, please start to delete the keys sequentially. During your process, show each intermediate tree diagram after every insertion.

- Label all node keys and children clearly.
- Highlight any splits that occur (show before-and-after).

For each split, briefly explain which node overflowed, what key was promoted, and how child pointers were redistributed. Once all keys have been inserted, show the final balanced 2–4 tree. Finally, for every deletion, explain how you maintained balance — e.g.

- When was a merge or rotation required?
- Which key moved down or up?
- Which node (if any) was removed?

Show the final tree (which should again be empty)

**********End**********