Strathmore
UNIVERSITY

Strathmore Institute of Management and Technology

Instructions

1. Only work in pairs.
2. Plagiarized work will only score 50% of the marks.
3. You have exactly one hour to complete this exercise.
4. This will count as your attendance.

Questions

(a) Explain the difference between a general Tree, Binary Tree and Binary Search Tree.

A binary search tree is where each node x stores an element such that the element stored in the left subtree of x are less than or equal to x and elements stores in the right subtree of x are greater than or equal to x whereas.

A binary tree is defined recusively. It consists of a root , left subtree and right subtree

A general tree is defined as a tree where each node may have zero or more children

(b) Draw a diagram to illustrate an array with at least 20 elements. Using this array, explain and demonstrate how:

i. A general tree can be created.
ii. A binary tree can be created.
iii. A binary search tree can be created.

(c) Using Java programming language, write a code to implement any tree and explain what is happening in every step using comments.

```
// Define a TreeNode class to represent the nodes of the binary search tree
class TreeNode {
   int data;
```

```
    TreeNode left;
    TreeNode right;

    public TreeNode(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

// Create a BinarySearchTree class for the tree operations
class BinarySearchTree {
    TreeNode root;

    public BinarySearchTree() {
        this.root = null;
    }

    // Insert a value into the binary search tree
    public void insert(int value) {
        root = insertRec(root, value);
    }

    private TreeNode insertRec(TreeNode root, int value) {
```

```
        if (root == null) {
            root = new TreeNode(value);
            return root;
        }
        if (value < root.data) {
            root.left = insertRec(root.left, value);
        } else if (value > root.data) {
            root.right = insertRec(root.right, value);
        }
        return root;
    }

    // In-order traversal function (left-root-right)
    public void inOrderTraversal(TreeNode node) {
        if (node != null) {
            inOrderTraversal(node.left);
            System.out.print(node.data + " ");
            inOrderTraversal(node.right);
        }
    }
}

public class BinarySearchTreeExample {
    public static void main(String[] args) {
```

```
        // Create a binary search tree
        BinarySearchTree tree = new BinarySearchTree();

        // Insert values into the tree
        tree.insert(10);
        tree.insert(5);
        tree.insert(15);
        tree.insert(3);
        tree.insert(7);

        // Display the tree structure
        // The tree should look like this:
        //      10
        //     /  \
        //    5   15
        //   / \
        //  3   7

        // Perform in-order traversal
        System.out.println("In-Order Traversal:");
        tree.inOrderTraversal(tree.root);
    }
}
```

(d) Explain 4 advantages of a BST.

Efficient searching: their structures that searching for a specific element can be done in time.

Sorted data: The elements in the tree are inherently sorted.

Space: They are memory efficient compared to other data structures.

Range queries: You can efficiently find all elements within a specific range.

(e) Explain 2 disadvantages of a BST.

Unbalanced trees: BSTs can be unbalanced over time leading to performance issues.

Inefficient for sorted data: If data is already sorted and inserted into BSTs it can lead to the worst case scenarios of completely unbalanced tree.