Assignment 3
Submission Date: 26/09/2025

ANSWERS

Q1. Bubble Sort
Write a program to implement Bubble Sort on an integer array.
Test Cases:
• Input: [5, 2, 9, 1, 5, 6] → Output: [1, 2, 5, 5, 6, 9]
• Input: [3, 2, 1] → Output: [1, 2, 3]
• Input: [1, 2, 3] → Output: [1, 2, 3]

```java
import java.util.Arrays;

public class BubbleSort {
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    Run | Debug
    public static void main(String[] args) {
        int[] arr1 = {5, 2, 9, 1, 5, 6};
        bubbleSort(arr1);
        System.out.println(Arrays.toString(arr1));

        int[] arr2 = {3, 2, 1};
        bubbleSort(arr2);
        System.out.println(Arrays.toString(arr2));

        int[] arr3 = {1, 2, 3};
        bubbleSort(arr3);
        System.out.println(Arrays.toString(arr3));
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac BubbleSort.java
PS C:\Users\baenu\Test\ADS Assignment 3> java BubbleSort
[1, 2, 5, 5, 6, 9]
[1, 2, 3]
[1, 2, 3]
```

Q2. Insertion Sort
Implement Insertion Sort to arrange elements in ascending order.
Test Cases:
• Input: [12, 11, 13, 5, 6] → Output: [5, 6, 11, 12, 13]
• Input: [4, 3, 2, 10, 12] → Output: [2, 3, 4, 10, 12]

```java
import java.util.Arrays;

public class InsertionSort {
    public static void insertionSort(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }

    Run | Debug
    public static void main(String[] args) {
        int[] arr1 = {12, 11, 13, 5, 6};
        insertionSort(arr1);
        System.out.println(Arrays.toString(arr1));

        int[] arr2 = {4, 3, 2, 10, 12};
        insertionSort(arr2);
        System.out.println(Arrays.toString(arr2));
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac InsertionSort.java
PS C:\Users\baenu\Test\ADS Assignment 3> java InsertionSort
[5, 6, 11, 12, 13]
[2, 3, 4, 10, 12]
```

Q3. Selection Sort
Sort an array using Selection Sort.
Test Cases:
• Input: [64, 25, 12, 22, 11] → Output: [11, 12, 22, 25, 64]
• Input: [29, 10, 14, 37, 13] → Output: [10, 13, 14, 29, 37]

```java
import java.util.Arrays;

public class SelectionSort {
    public static void selectionSort(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }

    Run | Debug
    public static void main(String[] args) {
        int[] arr1 = {64, 25, 12, 22, 11};
        selectionSort(arr1);
        System.out.println(Arrays.toString(arr1));

        int[] arr2 = {29, 10, 14, 37, 13};
        selectionSort(arr2);
        System.out.println(Arrays.toString(arr2));
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac SelectionSort.java
PS C:\Users\baenu\Test\ADS Assignment 3> java SelectionSort
[11, 12, 22, 25, 64]
[10, 13, 14, 29, 37]
```

Q4. Merge Sort
Write a recursive program to implement Merge Sort.
Test Cases:
• Input: [38, 27, 43, 3, 9, 82, 10] → Output: [3, 9, 10, 27, 38, 43, 82]
• Input: [5, 4, 3, 2, 1] → Output: [1, 2, 3, 4, 5]

```java
import java.util.Arrays;

public class MergeSort {
    public static void mergeSort(int[] arr, int l, int r) {
        if (l < r) {
            int m = (l + r) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    public static void merge(int[] arr, int l, int m, int r) {
        int n1 = m - l + 1;
        int n2 = r - m;
        int[] L = new int[n1];
        int[] R = new int[n2];
        for (int i = 0; i < n1; i++) L[i] = arr[l + i];
        for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];
        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) arr[k++] = L[i++];
            else arr[k++] = R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    Run | Debug
    public static void main(String[] args) {
        int[] arr1 = {38, 27, 43, 3, 9, 82, 10};
        mergeSort(arr1, l:0, arr1.length - 1);
        System.out.println(Arrays.toString(arr1));

        int[] arr2 = {5, 4, 3, 2, 1};
        mergeSort(arr2, l:0, arr2.length - 1);
        System.out.println(Arrays.toString(arr2));
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac MergeSort.java
PS C:\Users\baenu\Test\ADS Assignment 3> java MergeSort
[3, 9, 10, 27, 38, 43, 82]
[1, 2, 3, 4, 5]
```

Q5. Quick Sort
Implement Quick Sort using the last element as the pivot.
Test Cases:
• Input: [10, 7, 8, 9, 1, 5] → Output: [1, 5, 7, 8, 9, 10]
• Input: [1, 1, 1, 1] → Output: [1, 1, 1, 1]

```java
import java.util.Arrays;

public class QuickSort {
    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }

    Run | Debug
    public static void main(String[] args) {
        int[] arr1 = {10, 7, 8, 9, 1, 5};
        quickSort(arr1, low:0, arr1.length - 1);
        System.out.println(Arrays.toString(arr1));

        int[] arr2 = {1, 1, 1, 1};
        quickSort(arr2, low:0, arr2.length - 1);
        System.out.println(Arrays.toString(arr2));
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac QuickSort.java
PS C:\Users\baenu\Test\ADS Assignment 3> java QuickSort
[1, 5, 7, 8, 9, 10]
[1, 1, 1, 1]
```

Q6. Sorting Strings (Lexicographic Order)
Write a program to sort an array of strings using any sorting algorithm.
Test Cases:
• Input: ["apple", "banana", "cherry", "date"] → Output: ["apple", "banana", "cherry", "date"]
• Input: ["dog", "cat", "elephant", "bee"] → Output: ["bee", "cat", "dog", "elephant"]
Assignment: Tree Data Structures (Coding)

```java
import java.util.Arrays;

public class StringSort {
    Run | Debug
    public static void main(String[] args) {
        String[] arr1 = {"apple", "banana", "cherry", "date"};
        Arrays.sort(arr1);
        System.out.println(Arrays.toString(arr1));

        String[] arr2 = {"dog", "cat", "elephant", "bee"};
        Arrays.sort(arr2);
        System.out.println(Arrays.toString(arr2));
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac StringSort.java
PS C:\Users\baenu\Test\ADS Assignment 3> java StringSort
[apple, banana, cherry, date]
[bee, cat, dog, elephant]
```

Q7. Create a Binary Tree & Print Preorder Traversal
Write a program to create a binary tree and print its preorder traversal.
Test Case:
Tree:
1
/ \
2 3
/ \
4 5
Expected Output (Preorder): 1 2 4 5 3

```java
class Node {
    int data;
    Node left, right;
    Node(int data) { this.data = data; }
}

public class BinaryTreePreorder {
    public static void preorder(Node root) {
        if (root == null) return;
        System.out.print(root.data + " ");
        preorder(root.left);
        preorder(root.right);
    }

    Run | Debug
    public static void main(String[] args) {
        Node root = new Node(data:1);
        root.left = new Node(data:2);
        root.right = new Node(data:3);
        root.left.left = new Node(data:4);
        root.left.right = new Node(data:5);

        preorder(root);
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac BinaryTreePreorder.java
PS C:\Users\baenu\Test\ADS Assignment 3> java BinaryTreePreorder
1 2 4 5 3
```

Q8. Inorder, Preorder, Postorder Traversals
Implement recursive functions to print inorder, preorder, and postorder traversals.
Test Case:
Tree:
10
/ \
20 30
/ \
40 50
• Inorder: 40 20 50 10 30
• Preorder: 10 20 40 50 30
• Postorder: 40 50 20 30 10

```java
class NodeT {
    int data;
    NodeT left, right;
    NodeT(int data) { this.data = data; }
}

public class TreeTraversals {
    public static void inorder(NodeT root) {
        if (root == null) return;
        inorder(root.left);
        System.out.print(root.data + " ");
        inorder(root.right);
    }

    public static void preorder(NodeT root) {
        if (root == null) return;
        System.out.print(root.data + " ");
        preorder(root.left);
        preorder(root.right);
    }

    public static void postorder(NodeT root) {
        if (root == null) return;
        postorder(root.left);
        postorder(root.right);
        System.out.print(root.data + " ");
    }

    Run | Debug
    public static void main(String[] args) {
        NodeT root = new NodeT(data:10);
        root.left = new NodeT(data:20);
        root.right = new NodeT(data:30);
        root.left.left = new NodeT(data:40);
        root.left.right = new NodeT(data:50);

        System.out.print(s:"Inorder: ");
        inorder(root);
        System.out.println();

        System.out.print(s:"Preorder: ");
        preorder(root);
        System.out.println();

        System.out.print(s:"Postorder: ");
        postorder(root);
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac TreeTraversals.java
PS C:\Users\baenu\Test\ADS Assignment 3> java TreeTraversals
Inorder: 40 20 50 10 30
Preorder: 10 20 40 50 30
Postorder: 40 50 20 30 10
```

Q9. Level Order Traversal (BFS)
Implement level-order traversal of a binary tree.
Test Case:
Tree:
1
/ \
2 3
/ \ / \
4 5 6 7
Output: 1 2 3 4 5 6 7

```java
import java.util.LinkedList;
import java.util.Queue;

class NodeL {
    int data;
    NodeL left, right;
    NodeL(int data) { this.data = data; }
}

public class LevelOrderTraversal {
    public static void levelOrder(NodeL root) {
        if (root == null) return;
        Queue<NodeL> q = new LinkedList<>();
        q.add(root);
        while (!q.isEmpty()) {
            NodeL node = q.poll();
            System.out.print(node.data + " ");
            if (node.left != null) q.add(node.left);
            if (node.right != null) q.add(node.right);
        }
    }

    Run | Debug
    public static void main(String[] args) {
        NodeL root = new NodeL(data:1);
        root.left = new NodeL(data:2);
        root.right = new NodeL(data:3);
        root.left.left = new NodeL(data:4);
        root.left.right = new NodeL(data:5);
        root.right.left = new NodeL(data:6);
        root.right.right = new NodeL(data:7);

        levelOrder(root);
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac LevelOrderTraversal.java
PS C:\Users\baenu\Test\ADS Assignment 3> java LevelOrderTraversal
1 2 3 4 5 6 7
```

Q10. Insert into a BST
Write a program to insert nodes into a BST and print inorder traversal.
Test Case:
Insert: 50, 30, 20, 40, 70, 60, 80
Tree (BST) Inorder: 20 30 40 50 60 70 80

Search in BST:
Write a program to search for a value in a BST. Return true if found, false otherwise.
Test Case:
BST from Q4 → Search for 40 → Output: Found
BST from Q4 → Search for 90 → Output: Not Found

```java
class NodeBST {
    int data;
    NodeBST left, right;
    NodeBST(int data) { this.data = data; }
}

public class BSTOperations {
    public static NodeBST insert(NodeBST root, int key) {
        if (root == null) return new NodeBST(key);
        if (key < root.data) root.left = insert(root.left, key);
        else root.right = insert(root.right, key);
        return root;
    }

    public static void inorder(NodeBST root) {
        if (root != null) {
            inorder(root.left);
            System.out.print(root.data + " ");
            inorder(root.right);
        }
    }

    public static boolean search(NodeBST root, int key) {
        if (root == null) return false;
        if (root.data == key) return true;
        return key < root.data ? search(root.left, key) : search(root.right, key);
    }

    Run | Debug
    public static void main(String[] args) {
        NodeBST root = null;
        int[] values = {50, 30, 20, 40, 70, 60, 80};
        for (int v : values) root = insert(root, v);
        inorder(root);
        System.out.println();
        System.out.println(search(root, key:40) ? "Found" : "Not Found");
        System.out.println(search(root, key:90) ? "Found" : "Not Found");
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac BSTOperations.java
PS C:\Users\baenu\Test\ADS Assignment 3> java BSTOperations
20 30 40 50 60 70 80
Found
Not Found
```

Q11. Minimum & Maximum Node in BST
Find the smallest and largest element in a BST.
Test Case:
BST from Q4 → Min: 20, Max: 80

```java
class NodeMinMax {
    int data;
    NodeMinMax left, right;
    NodeMinMax(int data) { this.data = data; }
}

public class BSTMinMax {
    public static int findMin(NodeMinMax root) {
        while (root.left != null) root = root.left;
        return root.data;
    }

    public static int findMax(NodeMinMax root) {
        while (root.right != null) root = root.right;
        return root.data;
    }

    Run | Debug
    public static void main(String[] args) {
        NodeMinMax root = new NodeMinMax(data:50);
        root.left = new NodeMinMax(data:30);
        root.right = new NodeMinMax(data:70);
        root.left.left = new NodeMinMax(data:20);
        root.left.right = new NodeMinMax(data:40);
        root.right.left = new NodeMinMax(data:60);
        root.right.right = new NodeMinMax(data:80);

        System.out.println("Min: " + findMin(root));
        System.out.println("Max: " + findMax(root));
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac BSTMinMax.java
PS C:\Users\baenu\Test\ADS Assignment 3> java BSTMinMax
Min: 20
Max: 80
```

Q12. Height of Binary Tree
Write a recursive function to find the height of a binary tree.
Test Case:
Tree:
1
/ \
2 3
/ \
4 5
Height = 3

```java
class NodeH {
    int data;
    NodeH left, right;
    NodeH(int data) { this.data = data; }
}

public class BinaryTreeHeight {
    public static int height(NodeH root) {
        if (root == null) return 0;
        int lh = height(root.left);
        int rh = height(root.right);
        return 1 + Math.max(lh, rh);
    }

    Run | Debug
    public static void main(String[] args) {
        NodeH root = new NodeH(data:1);
        root.left = new NodeH(data:2);
        root.right = new NodeH(data:3);
        root.left.left = new NodeH(data:4);
        root.left.right = new NodeH(data:5);

        System.out.println("Height = " + height(root));
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac BinaryTreeHeight.java
PS C:\Users\baenu\Test\ADS Assignment 3> java BinaryTreeHeight
Height = 3
```

Q13. Check if a Binary Tree is Balanced
A balanced tree means the height difference of left and right subtrees for every node is ≤ 1.
Test Cases:
• Input: Balanced tree → Output: True
• Input: Skewed tree (like linked list: 1→2→3→4) → Output: False

```java
class NodeC {
    int data;
    NodeC left, right;
    NodeC(int data) { this.data = data; }
}

public class CheckBalancedTree {
    public static boolean isBalanced(NodeC root) {
        return checkHeight(root) != -1;
    }

    private static int checkHeight(NodeC root) {
        if (root == null) return 0;
        int lh = checkHeight(root.left);
        if (lh == -1) return -1;
        int rh = checkHeight(root.right);
        if (rh == -1) return -1;
        return Math.abs(lh - rh) > 1 ? -1 : Math.max(lh, rh) + 1;
    }

    Run | Debug
    public static void main(String[] args) {
        NodeC balanced = new NodeC(data:1);
        balanced.left = new NodeC(data:2);
        balanced.right = new NodeC(data:3);
        balanced.left.left = new NodeC(data:4);
        balanced.left.right = new NodeC(data:5);
        System.out.println(isBalanced(balanced));

        NodeC skewed = new NodeC(data:1);
        skewed.right = new NodeC(data:2);
        skewed.right.right = new NodeC(data:3);
        skewed.right.right.right = new NodeC(data:4);
        System.out.println(isBalanced(skewed));
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac CheckBalancedTree.java
PS C:\Users\baenu\Test\ADS Assignment 3> java CheckBalancedTree
true
false
```

Q14. Convert Sorted Array to Balanced BST
Write a program to convert a sorted array into a balanced BST.
Test Case:
Input: [1, 2, 3, 4, 5, 6, 7]
Output (Preorder example): 4 2 1 3 6 5 7

```java
class NodeS {
    int data;
    NodeS left, right;
    NodeS(int data) { this.data = data; }
}

public class SortedArrayToBST {
    public static NodeS sortedArrayToBST(int[] arr, int start, int end) {
        if (start > end) return null;
        int mid = (start + end) / 2;
        NodeS node = new NodeS(arr[mid]);
        node.left = sortedArrayToBST(arr, start, mid - 1);
        node.right = sortedArrayToBST(arr, mid + 1, end);
        return node;
    }

    public static void preorder(NodeS root) {
        if (root == null) return;
        System.out.print(root.data + " ");
        preorder(root.left);
        preorder(root.right);
    }

    Run | Debug
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5, 6, 7};
        NodeS root = sortedArrayToBST(arr, start:0, arr.length - 1);
        preorder(root);
    }
}
```

```
PS C:\Users\baenu\Test\ADS Assignment 3> javac SortedArrayToBST.java
PS C:\Users\baenu\Test\ADS Assignment 3> java SortedArrayToBST
4 2 1 3 6 5 7
```