1. Design and Develop Java Program to implement Polynomial addition using Arrays.

Program:-

```
import java.util.Scanner;
public class PolynomialAddition {
  // Method to add two polynomials
   public static int[] addPolynomials(int[] poly1, int[] poly2) {
      int length = Math.max(poly1.length, poly2.length);
      int[] result = new int[length];
      for (int i = 0; i < length; i++) {
         int coef1 = (i < poly1.length) ? poly1[i] : 0;
         int coef2 = (i < poly2.length) ? poly2[i] : 0;
         result[i] = coef1 + coef2;
      }
       return result;
   }
   // Method to display polynomial
   public static void displayPolynomial(int[] poly) {
      StringBuilder sb = new StringBuilder();
       for (int i = poly.length - 1; i >= 0; i--) {
         if (poly[i] != 0) {
            if (sb.length() > 0) {
               sb.append(" + ");
            }
            if (i == 0) {
               sb.append(poly[i]);
            } else if (i == 1) {
               sb.append(poly[i]).append("x");
            } else {
               sb.append(poly[i]).append("x^").append(i);
            }
         }
      }
            System.out.println(sb.length() > 0 ? sb.toString() : "0");
   }
      public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
            // Input for polynomial 1
      System.out.print("Enter the degree of the first polynomial: ");
      int degree1 = scanner.nextInt();
      int[] poly1 = new int[degree1 + 1];
```

```
System.out.println("Enter the coefficients for the first polynomial (from lowest to highest degree): ");
        for (int i = 0; i <= degree1; i++) {
            poly1[i] = scanner.nextInt();
        }
         // Input for polynomial 2
        System.out.print("Enter the degree of the second polynomial: ");
        int degree2 = scanner.nextInt();
        int[] poly2 = new int[degree2 + 1];
        System.out.println("Enter the coefficients for the second polynomial (from lowest to highest degree): ");
        for (int i = 0; i <= degree2; i++) {
            poly2[i] = scanner.nextInt();
        }
             // Add polynomials
        int[] result = addPolynomials(poly1, poly2);
          // Display results
        System.out.println("First Polynomial:");
        displayPolynomial(poly1);
        System.out.println("Second Polynomial:");
        displayPolynomial(poly2);
        System.out.println("Sum of Polynomials:");
        displayPolynomial(result);
         scanner.close();
    }
}
```

Out Put:

Enter the degree of the first polynomial:
2
Enter the coefficients for the first polynomial (from lowest to highest degree):
3
4
5
Enter the degree of the second polynomial:
1
Enter the coefficients for the second polynomial (from lowest to highest degree):
6
7
First Polynomial:
5x^2 + 4x + 3
Second Polynomial:
7x + 6
Sum of Polynomials:
5x^2 + 11x + 9

Conclusion:- The Polynomial addition using Arrays has been compiled and executed successfully.

2. Design and Develop menu driven Java Program to implement Stack and Queue operations using Single Linked List

Program:-

```
// Node class for linked list
class Node {
    int data;
    Node next;
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}
// Stack implementation using a single linked list
class Stack {
    private Node top;
    // Constructor
    public Stack() {
        this.top = null;
    }
    // Push operation
    public void push(int data) {
        Node newNode = new Node(data);
        newNode.next = top;
        top = newNode;
        System.out.println("Pushed " + data + " onto stack.");
    }
    // Pop operation
    public int pop() {
        if (isEmpty()) {
            throw new RuntimeException("Stack is empty. Cannot pop.");
        }
        int data = top.data;
        top = top.next;
        System.out.println("Popped " + data + " from stack.");
        return data;
    }
    // Peek operation
    public int peek() {
        if (isEmpty()) {
            throw new RuntimeException("Stack is empty. Cannot peek.");
        }
        return top.data;
```

```
    }

    // Check if stack is empty
    public boolean isEmpty() {
        return top == null;
    }
    // Print stack elements
    public void printStack() {
        Node temp = top;
        System.out.print("Stack: ");
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
}
// Queue implementation using a single linked list
class Queue {
    private Node front;
    private Node rear;
    // Constructor
    public Queue() {
        this.front = this.rear = null;
    }
    // Enqueue operation
    public void enqueue(int data) {
        Node newNode = new Node(data);
        if (rear == null) {
            front = rear = newNode;
        } else {
            rear.next = newNode;
            rear = newNode;
        }
        System.out.println("Enqueued " + data + " into queue.");
    }
    // Dequeue operation
```

```
    public int dequeue() {
        if (isEmpty()) {
            throw new RuntimeException("Queue is empty. Cannot dequeue.");
        }
        int data = front.data;
        front = front.next;
        if (front == null) {
            rear = null;
        }
        System.out.println("Dequeued " + data + " from queue.");
        return data;
    }
    // Peek operation
    public int peek() {
        if (isEmpty()) {
            throw new RuntimeException("Queue is empty. Cannot peek.");
        }
        return front.data;
    }
    // Check if queue is empty
    public boolean isEmpty() {
        return front == null;
    }
    // Print queue elements
    public void printQueue() {
        Node temp = front;
        System.out.print("Queue: ");
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
}
// Main class to test Stack and Queue
public class Main {
    public static void main(String[] args) {
        // Testing Stack
```

```
        System.out.println("Stack Operations:");
        Stack stack = new Stack();
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.printStack();
        stack.pop();
        stack.printStack();
        stack.peek();
        // Testing Queue
        System.out.println("\nQueue Operations:");
        Queue queue = new Queue();
        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        queue.printQueue();
        queue.dequeue();
        queue.printQueue();
        queue.peek();
    }
}
```

Output

Stack Operations:

Pushed 10 onto stack.

Pushed 20 onto stack.

Pushed 30 onto stack.

Stack: 30 20 10

Popped 30 from stack.

Stack: 20 10

20

Queue Operations:

Enqueued 10 into queue.

Enqueued 20 into queue.

Enqueued 30 into queue.

Queue: 10 20 30

Dequeued 10 from queue.

Queue: 20 30

20

Conclusion: The program has been compiled and executed successfully.

3. Design and Develop Java Program to implement for conversion expressions from infix to postfix and infix to prefix ?

Program:

```
import java.util.Stack;
public class ExpressionConverter {
   // Function to get precedence of operators
   private static int precedence(char op) {
      switch (op) {
         case '+':
         case '-':
            return 1;
         case '*':
         case '/':
            return 2;
         default:
            return -1;
      }
   }
   // Function to convert infix expression to postfix
   public static String infixToPostfix(String infix) {
      Stack<Character> stack = new Stack<>();
      StringBuilder postfix = new StringBuilder();
      for (int i = 0; i < infix.length(); i++) {
         char ch = infix.charAt(i);
         // If the character is an operand, add it to postfix expression
         if (Character.isLetterOrDigit(ch)) {
            postfix.append(ch);
         } else if (ch == '(') {
            stack.push(ch);
         } else if (ch == ')') {
            while (!stack.isEmpty() && stack.peek() != '(') {
               postfix.append(stack.pop());
            }
            stack.pop(); // Pop '('
         } else {
            while (!stack.isEmpty() && precedence(stack.peek()) >= precedence(ch)) {
               postfix.append(stack.pop());
            }
            stack.push(ch);
```

```
            }
        }
        // Pop all remaining operators from the stack
        while (!stack.isEmpty()) {
            postfix.append(stack.pop());
        }
        return postfix.toString();
    }
    // Function to reverse a string
    private static String reverse(String str) {
        return new StringBuilder(str).reverse().toString();
    }
    // Function to convert infix expression to prefix
    public static String infixToPrefix(String infix) {
        // Reverse the infix expression and swap parentheses
        String reversedInfix = reverse(infix)
                .replace('(', '>')
                .replace(')', '(')
                .replace('>', ')');
        // Convert reversed infix to postfix
        String postfix = infixToPostfix(reversedInfix);
        // Reverse postfix to get prefix
        return reverse(postfix);
    }
    public static void main(String[] args) {
        String infix = "A+B*(C-D)/E";
        System.out.println("Infix Expression: " + infix);
        String postfix = infixToPostfix(infix);
        System.out.println("Postfix Expression: " + postfix);
        String prefix = infixToPrefix(infix);
        System.out.println("Prefix Expression: " + prefix);
    }
}
```

OUT PUT:

Infix Expression: A+B*(C-D)/E
Postfix Expression: ABCD-*E/+
Prefix Expression: +A*B/-CDE

Conclusion: The program has been compiled and executed successfully.

4. Design and Develop Java Program to implement Binary Tree traversals.

Program :

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;
// Class representing a node in the binary tree
class TreeNode {
    int value;
    TreeNode left, right;
    public TreeNode(int item) {
        value = item;
        left = right = null;
    }
}
// Class representing the binary tree and its traversals
public class BinaryTree {
    TreeNode root;
    // In-order Traversal
    void inOrderTraversal(TreeNode node) {
        if (node == null) {
            return;
        }
        inOrderTraversal(node.left);
        System.out.print(node.value + " ");
        inOrderTraversal(node.right);
    }
    // Pre-order Traversal
    void preOrderTraversal(TreeNode node) {
        if (node == null) {
            return;
        }
        System.out.print(node.value + " ");
        preOrderTraversal(node.left);
        preOrderTraversal(node.right);
    }
    // Post-order Traversal
    void postOrderTraversal(TreeNode node) {
        if (node == null) {
            return;
        }
        postOrderTraversal(node.left);
        postOrderTraversal(node.right);
```

```
    System.out.print(node.value + " ");
}
// Level-order Traversal
void levelOrderTraversal(TreeNode node) {
    if (node == null) {
        return;
    }
    Queue<TreeNode> queue = new LinkedList<>();
    queue.add(node);
    while (!queue.isEmpty()) {
        TreeNode current = queue.poll();
        System.out.print(current.value + " ");
        if (current.left != null) {
            queue.add(current.left);
        }
        if (current.right != null) {
            queue.add(current.right);
        }
    }
}
// Main method to test the tree and traversals
public static void main(String[] args) {
    BinaryTree tree = new BinaryTree();
    // Creating the following binary tree
    //         1
    //        / \
    //       2   3
    //      / \ / \
    //     4  5 6  7
    tree.root = new TreeNode(1);
    tree.root.left = new TreeNode(2);
    tree.root.right = new TreeNode(3);
    tree.root.left.left = new TreeNode(4);
    tree.root.left.right = new TreeNode(5);
    tree.root.right.left = new TreeNode(6);
    tree.root.right.right = new TreeNode(7);
    System.out.println("In-order Traversal:");
    tree.inOrderTraversal(tree.root);
    System.out.println();
    System.out.println("Pre-order Traversal:");
    tree.preOrderTraversal(tree.root);
    System.out.println();
```

```
        System.out.println("Post-order Traversal:");
        tree.postOrderTraversal(tree.root);
        System.out.println();
        System.out.println("Level-order Traversal:");
        tree.levelOrderTraversal(tree.root);
        System.out.println();
    }
}
```

OUT PUT

In-order Traversal:

4 2 5 1 6 3 7

Pre-order Traversal:

1 2 4 5 3 6 7

Post-order Traversal:

4 5 2 6 7 3 1

Level-order Traversal:

1 2 3 4 5 6 7

5. Design and Develop menu driven Java Program to implement Graph traversal techniques.

Program:

```
import java.util.*;
public class GraphTraversal {
   static class Graph {
      private Map<Integer, List<Integer>> adjacencyList = new HashMap<>();
      // Add edge to the graph
      public void addEdge(int source, int destination) {
         adjacencyList.computeIfAbsent(source, k -> new ArrayList<>()).add(destination);
         adjacencyList.computeIfAbsent(destination, k -> new ArrayList<>()).add(source); // For
undirected graph
      }
      // Perform Depth-First Search
      public void depthFirstSearch(int start) {
         Set<Integer> visited = new HashSet<>();
         dfsHelper(start, visited);
         System.out.println();
      }
      private void dfsHelper(int node, Set<Integer> visited) {
         if (visited.contains(node)) return;
         visited.add(node);
         System.out.print(node + " ");
         for (int neighbor : adjacencyList.getOrDefault(node, Collections.emptyList())) {
            if (!visited.contains(neighbor)) {
               dfsHelper(neighbor, visited);
            }
         }
      }
      // Perform Breadth-First Search
      public void breadthFirstSearch(int start) {
         Set<Integer> visited = new HashSet<>();
         Queue<Integer> queue = new LinkedList<>();
         visited.add(start);
         queue.add(start);
         while (!queue.isEmpty()) {
            int node = queue.poll();
            System.out.print(node + " ");
            for (int neighbor : adjacencyList.getOrDefault(node, Collections.emptyList())) {
               if (!visited.contains(neighbor)) {
                  visited.add(neighbor);
                  queue.add(neighbor);
               }
```

```
            }
        }
        System.out.println();
    }
    // Print the graph
    public void printGraph() {
        for (Map.Entry<Integer, List<Integer>> entry : adjacencyList.entrySet()) {
            System.out.print(entry.getKey() + ": ");
            for (int neighbor : entry.getValue()) {
                System.out.print(neighbor + " ");
            }
            System.out.println();
        }
    }
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    Graph graph = new Graph();
    while (true) {
        System.out.println("Menu:");
        System.out.println("1. Add Edge");
        System.out.println("2. Print Graph");
        System.out.println("3. Perform Depth-First Search (DFS)");
        System.out.println("4. Perform Breadth-First Search (BFS)");
        System.out.println("5. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                System.out.print("Enter source vertex: ");
                int source = scanner.nextInt();
                System.out.print("Enter destination vertex: ");
                int destination = scanner.nextInt();
                graph.addEdge(source, destination);
                break;
            case 2:
                graph.printGraph();
                break;
            case 3:
                System.out.print("Enter starting vertex for DFS: ");
                int dfsStart = scanner.nextInt();
                System.out.print("DFS traversal starting from " + dfsStart + ": ");
```

```
                graph.depthFirstSearch(dfsStart);
                break;
            case 4:
                System.out.print("Enter starting vertex for BFS: ");
                int bfsStart = scanner.nextInt();
                System.out.print("BFS traversal starting from " + bfsStart + ": ");
                graph.breadthFirstSearch(bfsStart);
                break;
            case 5:
                System.out.println("Exiting...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
}
```

OUT PUT:

```
Menu:
1. Add Edge
2. Print Graph
3. Perform Depth-First Search (DFS)
4. Perform Breadth-First Search (BFS)
5. Exit
Enter your choice: 1
Enter source vertex: 1
Enter destination vertex: 2

Menu:
1. Add Edge
2. Print Graph
3. Perform Depth-First Search (DFS)
4. Perform Breadth-First Search (BFS)
5. Exit
Enter your choice: 1
Enter source vertex: 1
Enter destination vertex: 3
Menu:
1. Add Edge
2. Print Graph
```

3. Perform Depth-First Search (DFS)
4. Perform Breadth-First Search (BFS)
5. Exit
Enter your choice: 1
Enter source vertex: 2
Enter destination vertex: 4
Menu:
1. Add Edge
2. Print Graph
3. Perform Depth-First Search (DFS)
4. Perform Breadth-First Search (BFS)
5. Exit
Enter your choice: 2
1: 2 3
2: 1 4
3: 1
4: 2
Menu:
1. Add Edge
2. Print Graph
3. Perform Depth-First Search (DFS)
4. Perform Breadth-First Search (BFS)
5. Exit
Enter your choice: 3
Enter starting vertex for DFS: 1
DFS traversal starting from 1: 1 2 4 3
Menu:
1. Add Edge
2. Print Graph
3. Perform Depth-First Search (DFS)
4. Perform Breadth-First Search (BFS)
5. Exit
Enter your choice: 4
Enter starting vertex for BFS: 1
BFS traversal starting from 1: 1 2 3 4
Menu:
1. Add Edge
2. Print Graph
3. Perform Depth-First Search (DFS)
4. Perform Breadth-First Search (BFS)
5. Exit

Conclusion: The program has been compiled and executed successfully.

6. Design and Develop Java Program to implement Binary Search tree operations

Program:-

```
class BinarySearchTree {
// Define the structure of the node
static class TreeNode {
    int value;
    TreeNode left, right;
    TreeNode(int value) {
        this.value = value;
        left = right = null;
    }
}
private TreeNode root;
// Constructor
BinarySearchTree() {
    root = null;
}
// Insertion method
public void insert(int value) {
    root = insertRec(root, value);
}
private TreeNode insertRec(TreeNode node, int value) {
    // If the tree is empty, create a new node
    if (node == null) {
        node = new TreeNode(value);
        return node;
    }
    // Otherwise, recur down the tree
    if (value < node.value) {
        node.left = insertRec(node.left, value);
    } else if (value > node.value) {
        node.right = insertRec(node.right, value);
    }
    // Return the (unchanged) node pointer
    return node;
}
// Deletion method
public void delete(int value) {
    root = deleteRec(root, value);
}
private TreeNode deleteRec(TreeNode root, int value) {
    // Base Case: If the tree is empty
```

```
    if (root == null) {
        return root;
    }
    // Otherwise, recur down the tree
    if (value < root.value) {
        root.left = deleteRec(root.left, value);
    } else if (value > root.value) {
        root.right = deleteRec(root.right, value);
    } else {
        // Node to be deleted found
        // Case 1: Node with only one child or no child
        if (root.left == null) {
            return root.right;
        } else if (root.right == null) {
            return root.left;
        }
        // Case 2: Node with two children: Get the inorder successor (smallest in the right subtree)
        root.value = minValue(root.right);
        // Delete the inorder successor
        root.right = deleteRec(root.right, root.value);
    }
    return root;
}
// Helper function to find the minimum value node in the given subtree
private int minValue(TreeNode root) {
    int minValue = root.value;
    while (root.left != null) {
        minValue = root.left.value;
        root = root.left;
    }
    return minValue;
}
// Search method
public boolean search(int value) {
    return searchRec(root, value);
}
private boolean searchRec(TreeNode root, int value) {
    // Base Cases: root is null or key is present at the root
    if (root == null) {
```

```
        return false;
    }
    if (root.value == value) {
        return true;
    }
    // Key is greater than root's key
    if (root.value < value) {
        return searchRec(root.right, value);
    }
    // Key is smaller than root's key
    return searchRec(root.left, value);
}
// Traversal methods
public void inOrder() {
    inOrderRec(root);
    System.out.println();
}
private void inOrderRec(TreeNode root) {
    if (root != null) {
        inOrderRec(root.left);
        System.out.print(root.value + " ");
        inOrderRec(root.right);
    }
}
public void preOrder() {
    preOrderRec(root);
    System.out.println();
}
private void preOrderRec(TreeNode root) {
    if (root != null) {
        System.out.print(root.value + " ");
        preOrderRec(root.left);
        preOrderRec(root.right);
    }
}
public void postOrder() {
    postOrderRec(root);
    System.out.println();
}
private void postOrderRec(TreeNode root) {
    if (root != null) {
        postOrderRec(root.left);
```

```
            postOrderRec(root.right);
            System.out.print(root.value + " ");
        }
    }
    // Main method for testing
    public static void main(String[] args) {
        BinarySearchTree bst = new BinarySearchTree();
        // Insert elements
        bst.insert(50);
        bst.insert(30);
        bst.insert(20);
        bst.insert(40);
        bst.insert(70);
        bst.insert(60);
        bst.insert(80);
        // Print in-order traversal
        System.out.print("In-order traversal: ");
        bst.inOrder();  // Expected Output: 20 30 40 50 60 70 80
        // Print pre-order traversal
        System.out.print("Pre-order traversal: ");
        bst.preOrder(); // Expected Output: 50 30 20 40 70 60 80
        // Print post-order traversal
        System.out.print("Post-order traversal: ");
        bst.postOrder(); // Expected Output: 20 40 30 60 80 70 50
        // Search for elements
        System.out.println("Searching for 40: " + bst.search(40)); // Expected Output: true
        System.out.println("Searching for 100: " + bst.search(100)); // Expected Output: false
        // Delete an element
        bst.delete(20);
        System.out.print("In-order traversal after deleting 20: ");
        bst.inOrder(); // Expected Output: 30 40 50 60 70 80
        bst.delete(30);
        System.out.print("In-order traversal after deleting 30: ");
        bst.inOrder(); // Expected Output: 40 50 60 70 80
    }
}
```

OUT PUT:

In-order traversal after deleting 20: 30 40 50 60 70 80
In-order traversal after deleting 30: 40 50 60 70 80
In-order traversal: 20 30 40 50 60 70 80
Pre-order traversal: 50 30 20 40 70 60 80
Post-order traversal: 20 40 30 60 80 70 50
Searching for 40: true
Searching for 100: false
In-order traversal after deleting 20: 30 40 50 60 70 80
In-order traversal after deleting 30: 40 50 60 70 80

7. Design and Develop Java Program to implement AVL Trees.

Program:

```
// AVL Tree Node
class AVLNode {
    int key, height;
    AVLNode left, right;
    AVLNode(int key) {
        this.key = key;
        this.height = 1;
    }
}
// AVL Tree Class
public class AVLTree {
    private AVLNode root;
    // Utility method to get the height of the node
    private int height(AVLNode node) {
        return node == null ? 0 : node.height;
    }
    // Utility method to get the balance factor of the node
    private int getBalance(AVLNode node) {
        return node == null ? 0 : height(node.left) - height(node.right);
    }
    // Right rotate the subtree rooted with y
    private AVLNode rightRotate(AVLNode y) {
        AVLNode x = y.left;
        AVLNode T2 = x.right;
        x.right = y;
        y.left = T2;
        y.height = Math.max(height(y.left), height(y.right)) + 1;
        x.height = Math.max(height(x.left), height(x.right)) + 1;
        return x;
    }
    // Left rotate the subtree rooted with x
    private AVLNode leftRotate(AVLNode x) {
        AVLNode y = x.right;
        AVLNode T2 = y.left;
        y.left = x;
        x.right = T2;
        x.height = Math.max(height(x.left), height(x.right)) + 1;
        y.height = Math.max(height(y.left), height(y.right)) + 1;
```

```
    return y;
}
// Insert a key into the AVL Tree
public void insert(int key) {
    root = insert(root, key);
}
private AVLNode insert(AVLNode node, int key) {
    if (node == null) {
        return new AVLNode(key);
    }
    if (key < node.key) {
        node.left = insert(node.left, key);
    } else if (key > node.key) {
        node.right = insert(node.right, key);
    } else {
        // Duplicate keys are not allowed in AVL Tree
        return node;
    }
    node.height = Math.max(height(node.left), height(node.right)) + 1;
    int balance = getBalance(node);
    if (balance > 1 && key < node.left.key) {
        return rightRotate(node);
    }
    if (balance < -1 && key > node.right.key) {
        return leftRotate(node);
    }
    if (balance > 1 && key > node.left.key) {
        node.left = leftRotate(node.left);
        return rightRotate(node);
    }
    if (balance < -1 && key < node.right.key) {
        node.right = rightRotate(node.right);
        return leftRotate(node);
    }
    return node;
}
```

```
    // Inorder traversal of the AVL tree
    public void inorder() {
        inorder(root);
        System.out.println();
    }
    private void inorder(AVLNode node) {
        if (node != null) {
            inorder(node.left);
            System.out.print(node.key + " ");
            inorder(node.right);
        }
    }
    // Main method to test the AVL Tree implementation
    public static void main(String[] args) {
        AVLTree tree = new AVLTree();
        int[] keys = {10, 20, 30, 15, 25, 5, 3};
        for (int key : keys) {
            tree.insert(key);
        }
        System.out.println("Inorder traversal of the AVL tree:");
        tree.inorder();
    }
}
```

OUTPUT

```
In order traversal of the AVL tree:
  3 5 10 15 20 25 30
```

Conclusion: The program has been compiled and executed successfully.

8. Design and Develop menu driven Java Program to for Quick sort and Insertion sort.

Program:-

```
import java.util.Scanner;
public class SortMenu {
   // Method for Quick Sort
   public static void quickSort(int[] array, int low, int high) {
      if (low < high) {
         int pi = partition(array, low, high);
         quickSort(array, low, pi - 1); // Before pi
         quickSort(array, pi + 1, high); // After pi
      }
   }
   // Partition method for Quick Sort
   private static int partition(int[] array, int low, int high) {
      int pivot = array[high];
      int i = (low - 1); // Index of smaller element
      for (int j = low; j < high; j++) {
         if (array[j] <= pivot) {
            i++;
            // Swap array[i] and array[j]
            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
         }
      }
      // Swap array[i + 1] and array[high] (or pivot)
      int temp = array[i + 1];
      array[i + 1] = array[high];
      array[high] = temp;
      return i + 1;
   }
   // Method for Insertion Sort
   public static void insertionSort(int[] array) {
      int n = array.length;
      for (int i = 1; i < n; i++) {
         int key = array[i];
         int j = i - 1;
```

```
        // Move elements of array[0..i-1], that are greater than key, to one position ahead
        while (j >= 0 && array[j] > key) {
            array[j + 1] = array[j];
            j = j - 1;
        }
        array[j + 1] = key;
    }
}
// Method to print array
private static void printArray(int[] array) {
    for (int i : array) {
        System.out.print(i + " ");
    }
    System.out.println();
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    while (true) {
        System.out.println("Sorting Menu:");
        System.out.println("1. Quick Sort");
        System.out.println("2. Insertion Sort");
        System.out.println("3. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();
        if (choice == 3) {
            break;
        }
        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();
        int[] array = new int[n];
        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            array[i] = scanner.nextInt();
        }
        switch (choice) {
            case 1:
```

```
                quickSort(array, 0, array.length - 1);
                System.out.println("Sorted array using Quick Sort:");
                printArray(array);
                break;
            case 2:
                insertionSort(array);
                System.out.println("Sorted array using Insertion Sort:");
                printArray(array);
                break;
            default:
                System.out.println("Invalid choice, please try again.");
        }
      }
      scanner.close();
   }
}
```

OUTPUT:

```
 Using Quick Sort
Sorting Menu:
1. Quick Sort
2. Insertion Sort
3. Exit
Enter your choice: 1
Enter the number of elements: 5
Enter the elements:
64
25
12
22
11
Sorted array using Quick Sort:
11 12 22 25 64
Using Insertion Sort
Sorting Menu:
1. Quick Sort
2. Insertion Sort
3. Exit
Enter your choice: 2
Enter the number of elements: 6
Enter the elements:
34
```

7
23
32
5
62
Sorted array using Insertion Sort:
5 7 23 32 34 62
Exiting the Program
Sorting Menu:
1. Quick Sort
2. Insertion Sort
3. Exit
Enter your choice: 3

Conclusion: The program has been compiled and executed successfully.

9. Design and Develop menu driven Java Program to for Heap sort and Merge sort.

Program:-

```
import java.util.Scanner;
public class SortMenu {
   // Main method
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      while (true) {
         System.out.println("Menu:");
         System.out.println("1. Heap Sort");
         System.out.println("2. Merge Sort");
         System.out.println("3. Exit");
         System.out.print("Choose an option (1/2/3): ");
         int choice = scanner.nextInt();
         switch (choice) {
            case 1:
               performHeapSort(scanner);
               break;
            case 2:
               performMergeSort(scanner);
               break;
            case 3:
               System.out.println("Exiting...");
               return;
            default:
               System.out.println("Invalid choice. Please choose 1, 2, or 3.");
         }
      }
   }
   private static void performHeapSort(Scanner scanner) {
      System.out.print("Enter the number of elements: ");
      int n = scanner.nextInt();
      int[] array = new int[n];
      System.out.println("Enter the elements:");
      for (int i = 0; i < n; i++) {
         array[i] = scanner.nextInt();
      }
      heapSort(array);
      System.out.println("Sorted array using Heap Sort:");
      printArray(array);
   }
```

```
private static void performMergeSort(Scanner scanner) {
    System.out.print("Enter the number of elements: ");
    int n = scanner.nextInt();
    int[] array = new int[n];
    System.out.println("Enter the elements:");
    for (int i = 0; i < n; i++) {
        array[i] = scanner.nextInt();
    }
    mergeSort(array, 0, array.length - 1);
    System.out.println("Sorted array using Merge Sort:");
    printArray(array);
}
// Heap Sort Implementation
private static void heapSort(int[] array) {
    int n = array.length;
    // Build max heap
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(array, n, i);
    }
    // Extract elements from heap one by one
    for (int i = n - 1; i >= 0; i--) {
        // Move current root to end
        int temp = array[0];
        array[0] = array[i];
        array[i] = temp;
        // call max heapify on the reduced heap
        heapify(array, i, 0);
    }
}
private static void heapify(int[] array, int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && array[left] > array[largest]) {
        largest = left;
    }
    if (right < n && array[right] > array[largest]) {
        largest = right;
    }
    if (largest != i) {
        int swap = array[i];
```

```
        array[i] = array[largest];
        array[largest] = swap;
        heapify(array, n, largest);
    }
}
// Merge Sort Implementation
private static void mergeSort(int[] array, int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSort(array, left, mid);
        mergeSort(array, mid + 1, right);
        merge(array, left, mid, right);
    }
}
private static void merge(int[] array, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int[] L = new int[n1];
    int[] R = new int[n2];
    for (int i = 0; i < n1; i++) {
        L[i] = array[left + i];
    }
    for (int j = 0; j < n2; j++) {
        R[j] = array[mid + 1 + j];
    }
    int i = 0, j = 0;
    int k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            array[k] = L[i];
            i++;
        } else {
            array[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        array[k] = L[i];
        i++;
        k++;
    }
```

```
        while (j < n2) {
            array[k] = R[j];
            j++;
            k++;
        }
    }
    private static void printArray(int[] array) {
        for (int num : array) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

OUT PUT

```
Menu:
1. Heap Sort
2. Merge Sort
3. Exit
Choose an option (1/2/3):
1
Enter the number of elements:
6
Enter the elements:
25
7
18
9
12
3
Sorted array using Heap Sort:
3 7 9 12 18 25

Menu:
1. Heap Sort
2. Merge Sort
3. Exit
Choose an option (1/2/3):
2
```

Enter the number of elements:
5
Enter the elements:
10
2
8
4
6
Sorted array using Merge Sort:
2 4 6 8 10

Menu:
1. Heap Sort
2. Merge Sort
3. Exit
Choose an option (1/2/3):
3
Exiting...

Conclusion:- The program has been compiled and executed successfully.

10. Design and Develop menu driven Java Program to implement Linear search and Binary Search.

Program:

```
import java.util.Arrays;
import java.util.InputMismatchException;
import java.util.Scanner;
public class SearchProgram {
   private static Scanner scanner = new Scanner(System.in);
   public static void main(String[] args) {
      while (true) {
         System.out.println("Menu:");
         System.out.println("1. Linear Search");
         System.out.println("2. Binary Search");
         System.out.println("3. Exit");
         System.out.print("Choose an option: ");
         int choice = getValidIntInput();
         switch (choice) {
            case 1:
               performLinearSearch();
               break;
            case 2:
               performBinarySearch();
               break;
            case 3:
               System.out.println("Exiting...");
               return;
            default:
               System.out.println("Invalid choice. Please select again.");
         }
      }
   }
   private static void performLinearSearch() {
      System.out.print("Enter the number of elements: ");
      int n = getValidIntInput();
      int[] array = new int[n];
      System.out.println("Enter the elements:");
      for (int i = 0; i < n; i++) {
         array[i] = getValidIntInput();
      }
      System.out.print("Enter the element to search for: ");
      int target = getValidIntInput();
```

```
    int index = linearSearch(array, target);
    if (index != -1) {
        System.out.println("Element found at index: " + index);
    } else {
        System.out.println("Element not found.");
    }
}
private static void performBinarySearch() {
    System.out.print("Enter the number of elements: ");
    int n = getValidIntInput();
    int[] array = new int[n];
    System.out.println("Enter the elements (sorted order):");
    for (int i = 0; i < n; i++) {
        array[i] = getValidIntInput();
    }
    System.out.print("Enter the element to search for: ");
    int target = getValidIntInput();
    int index = binarySearch(array, target);
    if (index != -1) {
        System.out.println("Element found at index: " + index);
    } else {
        System.out.println("Element not found.");
    }
}

private static int linearSearch(int[] array, int target) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == target) {
            return i;
        }
    }
    return -1;
}
private static int binarySearch(int[] array, int target) {
    int left = 0;
    int right = array.length - 1;
     while (left <= right) {
     int mid = left + (right - left) / 2;
     if (array[mid] == target) {
            return mid;
        }
```

```
            if (array[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return -1;
    }

 private static int getValidIntInput() {
        while (true) {
            try {
                return scanner.nextInt();
            } catch (InputMismatchException e) {
                System.out.println("Invalid input. Please enter an integer.");
                scanner.next(); // Clear the invalid input
            }
        }
    }
}
```

OUT PUT:

Linear Search

```
Menu:
1. Linear Search
2. Binary Search
3. Exit
Choose an option: 1
Enter the number of elements: 5
Enter the elements:
10
20
30
40
50
Enter the element to search for: 30
Element found at index: 2
```

Binary Search

```
Menu:
1. Linear Search
```

2. Binary Search
3. Exit
Choose an option: 2
Enter the number of elements: 6
Enter the elements (sorted order):
5
15
25
35
45
55
Enter the element to search for: 35
Element found at index: 3

Invalid Input Handling

Menu:
1. Linear Search
2. Binary Search
3. Exit
Choose an option: 2
Enter the number of elements: 4
Enter the elements (sorted order):
10
20
30
40
Enter the element to search for: abc
Invalid input. Please enter an integer.
Enter the element to search for: 25
Element not found.

Exit Program

Menu:
1. Linear Search
2. Binary Search
3. Exit
Choose an option: 3
Exiting...

Conclusion: The program has been compiled and executed successfully.