1) Write a menu driven program for Linear and Binary Search.

1.1) Linear Search

Code:

```
import java.util.Scanner;
public class LinearSearch {
    public static int linearSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) {
                return i;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        System.out.print("Enter the target value to search: ");
        int target = scanner.nextInt();
        int result = linearSearch(arr, target);
        if (result != -1) {
            System.out.println("Linear Search: Element " + target + " is at index " + result);
        } else {
            System.out.println("Linear Search: Element not found.");
        }
        scanner.close();
    }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\LinearSearch.java
PS D:\FYMCA\ADS> java LinearSearch
Enter the number of elements in the array: 3
Enter 3 elements:
12
11
13
Enter the target value to search: 11
Linear Search: Element 11 is at index 1
```

1.2) Binary Search:

```
import java.util.Scanner;
public class BinarySearch {
   public static int binarySearch(int[] arr, int target) {
      int low = 0, high = arr.length - 1;
      while (low <= high) {
         int mid = (low + high) / 2;
         if (arr[mid] == target) {
            return mid;
         } else if (arr[mid] < target) {
            low = mid + 1;
         } else {
            high = mid - 1;
         }
      }
      return -1;
   }
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the number of elements in the array: ");
      int n = scanner.nextInt();
      int[] arr = new int[n];
      System.out.println("Enter " + n + " sorted elements:");
      for (int i = 0; i < n; i++) {
         arr[i] = scanner.nextInt();
      }
      System.out.print("Enter the target value to search: ");
      int target = scanner.nextInt();
      int result = binarySearch(arr, target);
      if (result != -1) {
         System.out.println("Binary Search: Element " + target + " is at index " + result);
      } else {
```

```
            System.out.println("Binary Search: Element not found.");
        }
        scanner.close();
    }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\BinarySearch.java
PS D:\FYMCA\ADS> java BinarySearch
Enter the number of elements in the array: 4
Enter 4 sorted elements:
13
10
11
17
Enter the target value to search: 11
Binary Search: Element 11 is at index 2
```

2) Write a menu driven program for Bubble and Selection Sort

2.1) Bubble sort:

Code:

```
import java.util.Scanner;
public class BubbleSort {
   public static void bubbleSort(int[] arr) {
      int n = arr.length;
      for (int i = 0; i < n; 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;
            }
         }
      }
   }
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the number of elements in the array: ");
      int n = scanner.nextInt();
      int[] arr = new int[n];
      System.out.println("Enter " + n + " elements:");
      for (int i = 0; i < n; i++) {
         arr[i] = scanner.nextInt();
      }
      bubbleSort(arr);
      System.out.print("Bubble Sort: Sorted Array: ");
      for (int num : arr) {
         System.out.print(num + " ");
      }
      System.out.println();
      scanner.close();
   }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\BubbleSort.java
PS D:\FYMCA\ADS> java BubbleSort
Enter the number of elements in the array: 3
Enter 3 elements:
11
10
13
Bubble Sort: Sorted Array: 10 11 13
```

2.2) Selection Sort

Code:

```
import java.util.Scanner;
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;
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        selectionSort(arr);
        System.out.print("Selection Sort: Sorted Array: ");
        for (int num : arr) {
            System.out.print(num + " ");
        }
```

```
        System.out.println();
        scanner.close();
    }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\SelectionSort.java
PS D:\FYMCA\ADS> java SelectionSort
Enter the number of elements in the array: 4
Enter 4 elements:
13
17
10
11
Selection Sort: Sorted Array: 10 11 13 17
```

3) Write a menu driven program for Insertion and Shell Sort

3.1) Insertion Sort:

Code:

```
import java.util.Scanner;
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 = j - 1;
         }
         arr[j + 1] = key;
      }
   }

   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the number of elements in the array: ");
      int n = scanner.nextInt();
      int[] arr = new int[n];
      System.out.println("Enter " + n + " elements:");
      for (int i = 0; i < n; i++) {
         arr[i] = scanner.nextInt();
      }
      insertionSort(arr);
      System.out.print("Insertion Sort: Sorted Array: ");
      for (int num : arr) {
         System.out.print(num + " ");
      }
      System.out.println();
      scanner.close();
   }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\InsertionSort.java
PS D:\FYMCA\ADS> java InsertionSort
Enter the number of elements in the array: 4
Enter 4 elements:
10
9
11
13
Insertion Sort: Sorted Array: 9 10 11 13
```

3.2) Shell Sort

```
import java.util.Scanner;
public class ShellSortMenuDriven {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] array = null;
        int choice;
        do {
            System.out.println("\nMenu:");
            System.out.println("1. Input Array");
            System.out.println("2. Display Array");
            System.out.println("3. Sort Array using Shell Sort");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Enter the size of the array: ");
                    int size = scanner.nextInt();
                    array = new int[size];
                    System.out.println("Enter " + size + " elements:");
                    for (int i = 0; i < size; i++) {
                        array[i] = scanner.nextInt();
                    }
                    System.out.println("Array input complete.");
                    break;
                case 2:
                    if (array == null) {
                        System.out.println("Array is empty! Please input the array first.");
                    } else {
                        System.out.print("Array elements: ");
                        for (int num : array) {
                            System.out.print(num + " ");
                        }
                        System.out.println();
                    }
```

```
                break;
            case 3:
                if (array == null) {
                    System.out.println("Array is empty! Please input the array first.");
                } else {
                    shellSort(array);
                    System.out.println("Array sorted using Shell Sort.");
                }
                break;
            case 4:
                System.out.println("Exiting program. Goodbye!");
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    } while (choice != 4);

    scanner.close();
  }
  public static void shellSort(int[] array) {
    int n = array.length;
    for (int gap = n / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = array[i];
            int j;
            for (j = i; j >= gap && array[j - gap] > temp; j -= gap) {
                array[j] = array[j - gap];
            }
            array[j] = temp;
        }
    }
  }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\ShellSortMenuDriven.java
PS D:\FYMCA\ADS> java ShellSortMenuDriven

Menu:
1. Input Array
2. Display Array
3. Sort Array using Shell Sort
4. Exit
Enter your choice: 1
Enter the size of the array: 5
Enter 5 elements:
10
9
11
13
8
Array input complete.

Menu:
1. Input Array
2. Display Array
3. Sort Array using Shell Sort
4. Exit
Enter your choice: 2
Array elements: 10 9 11 13 8

Menu:
1. Input Array
2. Display Array
3. Sort Array using Shell Sort
4. Exit
Enter your choice: 3
Array sorted using Shell Sort.
```

4) Write a program to implement Stack using Array and Linked List

```
import java.util.Scanner;
class StackUsingArray {
    private int[] stack;
    private int top;
    private int capacity;
    public StackUsingArray(int size) {
        capacity = size;
        stack = new int[capacity];
        top = -1;
    }
    public void push(int value) {
        if (top == capacity - 1) {
            System.out.println("Stack Overflow!");
        } else {
            stack[++top] = value;
            System.out.println("Pushed " + value);
        }
    }
    public int pop() {
        if (top == -1) {
            System.out.println("Stack Underflow!");
            return -1;
        } else {
            return stack[top--];
        }
    }
    public void display() {
        if (top == -1) {
            System.out.println("Stack is empty!");
        } else {
            System.out.println("Stack elements:");
            for (int i = top; i >= 0; i--) {
                System.out.println(stack[i]);
            }
        }
    }
}
class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
```

```
}
class StackUsingLinkedList {
    private Node top;
    public StackUsingLinkedList() {
        top = null;
    }
    public void push(int value) {
        Node newNode = new Node(value);
        if (top == null) {
            top = newNode;
        } else {
            newNode.next = top;
            top = newNode;
        }
        System.out.println("Pushed " + value);
    }

    public int pop() {
        if (top == null) {
            System.out.println("Stack Underflow!");
            return -1;
        } else {
            int value = top.data;
            top = top.next;
            return value;
        }
    }
    public void display() {
        if (top == null) {
            System.out.println("Stack is empty!");
        } else {
            System.out.println("Stack elements:");
            Node temp = top;
            while (temp != null) {
                System.out.println(temp.data);
                temp = temp.next;
            }
        }
    }
}
public class StackImplementation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        StackUsingArray arrayStack = null;
        StackUsingLinkedList linkedListStack = new StackUsingLinkedList();

        int choice;
        do {
            System.out.println("\nMenu:");
```

```
System.out.println("1. Create Stack Using Array");
System.out.println("2. Push into Array Stack");
System.out.println("3. Pop from Array Stack");
System.out.println("4. Display Array Stack");
System.out.println("5. Push into Linked List Stack");
System.out.println("6. Pop from Linked List Stack");
System.out.println("7. Display Linked List Stack");
System.out.println("8. Exit");
System.out.print("Enter your choice: ");
choice = scanner.nextInt();
switch (choice) {
    case 1:
        System.out.print("Enter the size of the array stack: ");
        int size = scanner.nextInt();
        arrayStack = new StackUsingArray(size);
        System.out.println("Array Stack created with size " + size);
        break;
    case 2:
        if (arrayStack == null) {
            System.out.println("Create the Array Stack first!");
        } else {
            System.out.print("Enter value to push: ");
            int value = scanner.nextInt();
            arrayStack.push(value);
        }
        break;
    case 3:
        if (arrayStack == null) {
            System.out.println("Create the Array Stack first!");
        } else {
            int poppedValue = arrayStack.pop();
            if (poppedValue != -1) {
                System.out.println("Popped: " + poppedValue);
            }
        }
        break;
    case 4:
        if (arrayStack == null) {
            System.out.println("Create the Array Stack first!");
        } else {
            arrayStack.display();
        }
        break;
    case 5:
        System.out.print("Enter value to push: ");
        int value = scanner.nextInt();
        linkedListStack.push(value);
        break;
    case 6:
```

```
                int poppedValue = linkedListStack.pop();
                if (poppedValue != -1) {
                    System.out.println("Popped: " + poppedValue);
                }
                break;
            case 7:
                linkedListStack.display();
                break;
            case 8:
                System.out.println("Exiting program. Goodbye!");
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    } while (choice != 8);

    scanner.close();
  }
}
```

Output

```
PS D:\FYMCA\ADS> javac .\StackImplementation.java
PS D:\FYMCA\ADS> java StackImplementation

Menu:
1. Create Stack Using Array
2. Push into Array Stack
3. Pop from Array Stack
4. Display Array Stack
5. Push into Linked List Stack
6. Pop from Linked List Stack
7. Display Linked List Stack
8. Exit
Enter your choice: 1
Enter the size of the array stack: 5
Array Stack created with size 5

Menu:
1. Create Stack Using Array
2. Push into Array Stack
3. Pop from Array Stack
4. Display Array Stack
5. Push into Linked List Stack
6. Pop from Linked List Stack
7. Display Linked List Stack
8. Exit
Enter your choice: 2
Enter value to push: 9
Pushed 9

Menu:
1. Create Stack Using Array
2. Push into Array Stack
3. Pop from Array Stack
4. Display Array Stack
5. Push into Linked List Stack
6. Pop from Linked List Stack
7. Display Linked List Stack
8. Exit
Enter your choice: 5
Enter value to push: 10
Pushed 10

Menu:
1. Create Stack Using Array
2. Push into Array Stack
3. Pop from Array Stack
4. Display Array Stack
5. Push into Linked List Stack
```

5) Write a program to implement Queue using LinkedList

Code:

```
import java.util.Scanner;
class Node {
    int data;
    Node next;
    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}
class QueueUsingLinkedList {
    private Node front, rear;
    public QueueUsingLinkedList() {
        front = rear = null;
    }
    public void enqueue(int value) {
        Node newNode = new Node(value);
        if (rear == null) {
            front = rear = newNode;
        } else {
            rear.next = newNode;
            rear = newNode;
        }
        System.out.println("Enqueued: " + value);
    }
    public int dequeue() {
        if (front == null) {
            System.out.println("Queue Underflow! The queue is empty.");
            return -1;
        } else {
            int value = front.data;
            front = front.next;
            if (front == null) {
                rear = null;
            }
            return value;
        }
    }
    public void display() {
        if (front == null) {
            System.out.println("Queue is empty!");
        } else {
            System.out.print("Queue elements: ");
            Node temp = front;
            while (temp != null) {
                System.out.print(temp.data + " ");
```

```
            temp = temp.next;
        }
        System.out.println();
    }
  }
}
public class QueueImplementation {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    QueueUsingLinkedList queue = new QueueUsingLinkedList();
    int choice;
    do {
      System.out.println("\nMenu:");
      System.out.println("1. Enqueue");
      System.out.println("2. Dequeue");
      System.out.println("3. Display Queue");
      System.out.println("4. Exit");
      System.out.print("Enter your choice: ");
      choice = scanner.nextInt();
      switch (choice) {
        case 1:
          System.out.print("Enter value to enqueue: ");
          int value = scanner.nextInt();
          queue.enqueue(value);
          break;
        case 2:
          int dequeuedValue = queue.dequeue();
          if (dequeuedValue != -1) {
            System.out.println("Dequeued: " + dequeuedValue);
          }
          break;
        case 3:
          queue.display();
          break;
        case 4:
          System.out.println("Exiting program. Goodbye!");
          break;
        default:
          System.out.println("Invalid choice. Please try again.");
      }
    } while (choice != 4);
    scanner.close();
  }
}
```

Output:

```
Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
Enqueued: 10

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter value to enqueue: 20
Enqueued: 20

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 3
Queue elements: 10 20

Menu:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 4
Exiting program. Goodbye!
```

6) Write a program to implement Functions on LinkedList (Creation, Deletion, Addition of Node, Searching, Traversal)

```
import java.util.Scanner;
class Node {
    int data;
    Node next;
    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}
class LinkedList {
    private Node head;
    public LinkedList() {
        this.head = null;
    }
    public void createNode(int value) {
        Node newNode = new Node(value);
        if (head == null) {
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
```

```
        }
        System.out.println("Node with value " + value + " created.");
    }
    public void deleteNode(int value) {
        if (head == null) {
            System.out.println("List is empty. No nodes to delete.");
            return;
        }
        if (head.data == value) {
            head = head.next;
            System.out.println("Node with value " + value + " deleted.");
            return;
        }
        Node temp = head;
        while (temp.next != null && temp.next.data != value) {
            temp = temp.next;
        }
        if (temp.next == null) {
            System.out.println("Node with value " + value + " not found.");
        } else {
            temp.next = temp.next.next;
            System.out.println("Node with value " + value + " deleted.");
        }
    }

    public void addNodeAtPosition(int value, int position) {
        Node newNode = new Node(value);
        if (position == 1) {
            newNode.next = head;
            head = newNode;
            System.out.println("Node with value " + value + " added at position " + position
+ ".");
            return;
        }
        Node temp = head;
        for (int i = 1; i < position - 1 && temp != null; i++) {
            temp = temp.next;
        }
        if (temp == null) {
            System.out.println("Position " + position + " is invalid.");
        } else {
            newNode.next = temp.next;
            temp.next = newNode;
            System.out.println("Node with value " + value + " added at position " + position
+ ".");
        }
    public void searchNode(int value) {
        Node temp = head;
        int position = 1;
```

```
        while (temp != null) {
            if (temp.data == value) {
                System.out.println("Node with value " + value + " found at position " +
position + ".");
                return;
            }
            temp = temp.next;
            position++;
        }
        System.out.println("Node with value " + value + " not found.");
    }
    public void traverse() {
        if (head == null) {
            System.out.println("List is empty!");
            return;
        }
        System.out.print("Linked List elements: ");
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
}
public class LinkedListFunctions {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        LinkedList list = new LinkedList();

        int choice;
        do {
            System.out.println("\nMenu:");
            System.out.println("1. Create Node (Add at End)");
            System.out.println("2. Delete Node by Value");
            System.out.println("3. Add Node at Specific Position");
            System.out.println("4. Search Node by Value");
            System.out.println("5. Traverse Linked List");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter value to create node: ");
                    int createValue = scanner.nextInt();
                    list.createNode(createValue);
                    break;
                case 2:
```

```
                System.out.print("Enter value to delete: ");
                int deleteValue = scanner.nextInt();
                list.deleteNode(deleteValue);
                break;
            case 3:
                System.out.print("Enter value to add: ");
                int addValue = scanner.nextInt();
                System.out.print("Enter position to add the node: ");
                int position = scanner.nextInt();
                list.addNodeAtPosition(addValue, position);
                break;
            case 4:
                System.out.print("Enter value to search: ");
                int searchValue = scanner.nextInt();
                list.searchNode(searchValue);
                break;
            case 5:
                list.traverse();
                break;
            case 6:
                System.out.println("Exiting program. Goodbye!");
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    } while (choice != 6);
    scanner.close();
  }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\LinkedListFunctions.java
PS D:\FYMCA\ADS> java LinkedListFunctions

Menu:
1. Create Node (Add at End)
2. Delete Node by Value
3. Add Node at Specific Position
4. Search Node by Value
5. Traverse Linked List
6. Exit
Enter your choice: 1
Enter value to create node: 10
Node with value 10 created.

Menu:
1. Create Node (Add at End)
2. Delete Node by Value
3. Add Node at Specific Position
4. Search Node by Value
5. Traverse Linked List
6. Exit
Enter your choice: 1
Enter value to create node: 20
Node with value 20 created.

Menu:
1. Create Node (Add at End)
2. Delete Node by Value
3. Add Node at Specific Position
4. Search Node by Value
5. Traverse Linked List
6. Exit
Enter your choice: 5
Linked List elements: 10 20

Menu:
1. Create Node (Add at End)
2. Delete Node by Value
3. Add Node at Specific Position
4. Search Node by Value
5. Traverse Linked List
6. Exit
Enter your choice: 3
Enter value to add: 15
Enter position to add the node: 2
Node with value 15 added at position 2.
```

7) Write a program to convert Infix to Postfix expression using Stack.

Code:

```
import java.util.Stack;
public class InfixToPostfix {
    private static boolean isOperator(char c) {
        return c == '+' || c == '-' || c == '*' || c == '/' || c == '^';
    }
    private static int precedence(char c) {
        switch (c) {
            case '+':
            case '-':
                return 1;
            case '*':
            case '/':
                return 2;
            case '^':
                return 3;
            default:
                return -1;
        }
    }
    public static String infixToPostfix(String infix) {
        StringBuilder postfix = new StringBuilder();
        Stack<Character> stack = new Stack<>();
        for (int i = 0; i < infix.length(); i++) {
            char c = infix.charAt(i);
            if (Character.isLetterOrDigit(c)) {
                postfix.append(c);
            }
            else if (c == '(') {
                stack.push(c);
            }
            else if (c == ')') {
                while (!stack.isEmpty() && stack.peek() != '(') {
                    postfix.append(stack.pop());
                }
                if (!stack.isEmpty() && stack.peek() == '(') {
                    stack.pop();
                } else {
                    System.out.println("Invalid expression");
                    return null;
                }
```

```
            }
            else if (isOperator(c)) {
                while (!stack.isEmpty() && precedence(stack.peek()) >= precedence(c)) {
                    postfix.append(stack.pop());
                }
                stack.push(c);
            }
        }
        while (!stack.isEmpty()) {
            if (stack.peek() == '(') {
                System.out.println("Invalid expression");
                return null;
            }
            postfix.append(stack.pop());
        }
        return postfix.toString();
    }
    public static void main(String[] args) {
        String infixExpression = "a+b*(c^d-e)^(f+g*h)-i";
        System.out.println("Infix Expression: " + infixExpression);
        String postfixExpression = infixToPostfix(infixExpression);
        if (postfixExpression != null) {
            System.out.println("Postfix Expression: " + postfixExpression);
        }
    }
}
```

Output:

```
PS D:\FYMCA\ADS> javac InfixToPostfix.java
PS D:\FYMCA\ADS> java InfixToPostfix.java
Infix Expression: a+b*(c^d-e)^(f+g*h)-i
Postfix Expression: abcd^e-fgh*+^*+i-
PS D:\FYMCA\ADS>
```

8) Write a program to evaluate the Postfix expression using Stack.

Code:

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import java.util.Stack;
public class PostfixEvaluation {
   public static int evaluatePostfix(String expression, Map<Character, Integer>
operandValues) {
      Stack<Integer> stack = new Stack<>();
      for (int i = 0; i < expression.length(); i++) {
         char c = expression.charAt(i);
         if (Character.isLetterOrDigit(c)) {
            if (operandValues.containsKey(c)) {
               stack.push(operandValues.get(c));
            } else {
               throw new IllegalArgumentException("Operand value for '" + c + "' not
provided.");
            }
         }
         else {
            int b = stack.pop();
            int a = stack.pop();
            switch (c) {
               case '+':
                  stack.push(a + b);
                  break;
               case '-':
                  stack.push(a - b);
                  break;
               case '*':
                  stack.push(a * b);
                  break;
               case '/':
                  if (b == 0) {
                     throw new ArithmeticException("Division by zero.");
                  }
                  stack.push(a / b);
                  break;
               case '^':
                  stack.push((int) Math.pow(a, b));
                  break;
               default:
                  throw new IllegalArgumentException("Invalid operator: " + c);
            }
```

```
            }
        }
        return stack.pop();
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a postfix expression: ");
        String postfixExpression = scanner.nextLine();
        Map<Character, Integer> operandValues = new HashMap<>();
        for (char c : postfixExpression.toCharArray()) {
            if (Character.isLetter(c) && !operandValues.containsKey(c)) {
                System.out.print("Enter value for " + c + ": ");
                operandValues.put(c, scanner.nextInt());
            }
        }
        try {
            int result = evaluatePostfix(postfixExpression, operandValues);
            System.out.println("The result of the postfix evaluation is: " + result);
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\PostfixEvaluation.java
PS D:\FYMCA\ADS> java PostfixEvaluation
Enter a postfix expression: ab+c*
Enter value for a: 2
Enter value for b: 3
Enter value for c: 4
The result of the postfix evaluation is: 20
```

9) Write a program for balancing the parentheses using Stack.

Code:

```
import java.util.Scanner;
import java.util.Stack;
public class ParenthesesBalancer {
    public static boolean isBalanced(String expression) {
        Stack<Character> stack = new Stack<>();
        for (char c : expression.toCharArray()) {
            if (c == '(' || c == '{' || c == '[') {
                stack.push(c);
            }
            else if (c == ')' || c == '}' || c == ']') {
                if (stack.isEmpty() || !isMatchingPair(stack.pop(), c)) {
                    return false;
                }
            }
        }
        return stack.isEmpty();
    }
    private static boolean isMatchingPair(char open, char close) {
        return (open == '(' && close == ')') ||
               (open == '{' && close == '}') ||
               (open == '[' && close == ']');
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an expression with parentheses: ");
        String expression = scanner.nextLine();
        if (isBalanced(expression)) {
            System.out.println("The parentheses are balanced.");
        } else {
            System.out.println("The parentheses are not balanced.");
        }
        scanner.close();
    }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\ParenthesesBalancer.java
PS D:\FYMCA\ADS> java ParenthesesBalancer
Enter an expression with parentheses: {[(a+b)*c]-d}
The parentheses are balanced.
```

MRIGANKA ADHIKARY
ROLL NO: 69

10) Write a program to implement Circular Queue.

Code:

```
import java.util.Scanner;
class CircularQueue {
    private int[] queue;
    private int front, rear, size, capacity;
    public CircularQueue(int capacity) {
        this.capacity = capacity;
        queue = new int[capacity];
        front = -1;
        rear = -1;
        size = 0;
    }
    public boolean isEmpty() {
        return size == 0;
    }
    public boolean isFull() {
        return size == capacity;
    }
    public void enqueue(int value) {
        if (isFull()) {
            System.out.println("Queue is full. Cannot enqueue " + value);
            return;
        }
        if (front == -1) {
            front = 0;
        }
        rear = (rear + 1) % capacity;
        queue[rear] = value;
        size++;
        System.out.println(value + " enqueued.");
    }
    public int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty. Cannot dequeue.");
            return -1;
        }
        int value = queue[front];
        front = (front + 1) % capacity;
        size--;
        return value;
    }
    public int peek() {
        if (isEmpty()) {
            System.out.println("Queue is empty. No front element.");
            return -1;
        }
```

```
        return queue[front];
    }
    public void display() {
        if (isEmpty()) {
            System.out.println("Queue is empty.");
            return;
        }
        System.out.print("Queue elements: ");
        for (int i = 0; i < size; i++) {
            System.out.print(queue[(front + i) % capacity] + " ");
        }
        System.out.println();
    }
}
public class CircularQueueDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the capacity of the circular queue: ");
        int capacity = scanner.nextInt();
        CircularQueue queue = new CircularQueue(capacity);
        while (true) {
            System.out.println("\nMenu:");
            System.out.println("1. Enqueue");
            System.out.println("2. Dequeue");
            System.out.println("3. Peek");
            System.out.println("4. Display");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Enter a value to enqueue: ");
                    int value = scanner.nextInt();
                    queue.enqueue(value);
                    break;
                case 2:
                    int dequeuedValue = queue.dequeue();
                    if (dequeuedValue != -1) {
                        System.out.println("Dequeued value: " + dequeuedValue);
                    }
                    break;
                case 3:
                    int frontValue = queue.peek();
                    if (frontValue != -1) {
                        System.out.println("Front value: " + frontValue);
                    }
                    break;
                case 4:
                    queue.display();
```

```
                break;
            case 5:
                System.out.println("Exiting...");
                scanner.close();
                System.exit(0);
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
  }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\CircularQueueDemo.java
PS D:\FYMCA\ADS> java CircularQueueDemo
Enter the capacity of the circular queue: 3

Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter a value to enqueue: 10
10 enqueued.

Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter a value to enqueue: 20
20 enqueued.

Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Queue elements: 10 20
```

11) Write a program to implement Priority Queue.

Code:

```
import java.util.Scanner;
class PriorityQueue {
    private int[] elements;
    private int[] priorities;
    private int size;
    private int capacity;
    public PriorityQueue(int capacity) {
        this.capacity = capacity;
        elements = new int[capacity];
        priorities = new int[capacity];
        size = 0;
    }
    public boolean isEmpty() {
        return size == 0;
    }
    public boolean isFull() {
        return size == capacity;
    }
    public void enqueue(int value, int priority) {
        if (isFull()) {
            System.out.println("Priority Queue is full. Cannot enqueue " + value);
            return;
        }
        int i;
        for (i = size - 1; i >= 0 && priorities[i] > priority; i--) {
            elements[i + 1] = elements[i];
            priorities[i + 1] = priorities[i];
        }
        elements[i + 1] = value;
        priorities[i + 1] = priority;
        size++;
        System.out.println(value + " enqueued with priority " + priority + ".");
    }
    public int dequeue() {
        if (isEmpty()) {
            System.out.println("Priority Queue is empty. Cannot dequeue.");
            return -1;
        }
        int value = elements[0];
        for (int i = 1; i < size; i++) {
            elements[i - 1] = elements[i];
            priorities[i - 1] = priorities[i];
        }
        size--;
        return value;
```

```
    }
    public void display() {
        if (isEmpty()) {
            System.out.println("Priority Queue is empty.");
            return;
        }
        System.out.println("Elements in Priority Queue:");
        for (int i = 0; i < size; i++) {
            System.out.println("Value: " + elements[i] + ", Priority: " + priorities[i]);
        }
    }
}
public class PriorityQueueDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the capacity of the Priority Queue: ");
        int capacity = scanner.nextInt();
        PriorityQueue pq = new PriorityQueue(capacity);
        while (true) {
            System.out.println("\nMenu:");
            System.out.println("1. Enqueue");
            System.out.println("2. Dequeue");
            System.out.println("3. Display");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Enter value to enqueue: ");
                    int value = scanner.nextInt();
                    System.out.print("Enter its priority: ");
                    int priority = scanner.nextInt();
                    pq.enqueue(value, priority);
                    break;
                case 2:
                    int dequeuedValue = pq.dequeue();
                    if (dequeuedValue != -1) {
                        System.out.println("Dequeued value: " + dequeuedValue);
                    }
                    break;
                case 3:
                    pq.display();
                    break;
                case 4:
                    System.out.println("Exiting...");
                    scanner.close();
                    System.exit(0);
                default:
                    System.out.println("Invalid choice. Please try again.");
```

```
            }
        }
    }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\PriorityQueueDemo.java
PS D:\FYMCA\ADS> java PriorityQueueDemo
Enter the capacity of the Priority Queue: 3

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
Enter its priority: 2
10 enqueued with priority 2.

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 20
Enter its priority: 1
20 enqueued with priority 1.

Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Elements in Priority Queue:
Value: 20, Priority: 1
Value: 10, Priority: 2
```

12) Write a program to implement Double ended Queue.

Code:

```
import java.util.Scanner;
class Deque {
    private int[] deque;
    private int front, rear, size, capacity;
    public Deque(int capacity) {
        this.capacity = capacity;
        deque = new int[capacity];
        front = -1;
        rear = -1;
        size = 0;
    }
    public boolean isEmpty() {
        return size == 0;
    }
    public boolean isFull() {
        return size == capacity;
    }
    public void insertFront(int value) {
        if (isFull()) {
            System.out.println("Deque is full. Cannot insert " + value + " at the front.");
            return;
        }
        if (front == -1) {
            front = 0;
            rear = 0;
        } else {
            front = (front - 1 + capacity) % capacity;
        }
        deque[front] = value;
        size++;
        System.out.println(value + " inserted at the front.");
    }
    public void insertRear(int value) {
        if (isFull()) {
            System.out.println("Deque is full. Cannot insert " + value + " at the rear.");
            return;
        }
        if (front == -1) {
            front = 0;
            rear = 0;
        } else {
            rear = (rear + 1) % capacity;
        }
        deque[rear] = value;
        size++;
```

```
        System.out.println(value + " inserted at the rear.");
    }
    public int deleteFront() {
        if (isEmpty()) {
            System.out.println("Deque is empty. Cannot delete from front.");
            return -1;
        }
        int value = deque[front];
        if (front == rear) {
            front = rear = -1;
        } else {
            front = (front + 1) % capacity;
        }
        size--;
        return value;
    }
    public int deleteRear() {
        if (isEmpty()) {
            System.out.println("Deque is empty. Cannot delete from rear.");
            return -1;
        }
        int value = deque[rear];
        if (front == rear) {
            front = rear = -1;
        } else {
            rear = (rear - 1 + capacity) % capacity;
        }
        size--;
        return value;
    }
    public void display() {
        if (isEmpty()) {
            System.out.println("Deque is empty.");
            return;
        }
        System.out.print("Deque elements: ");
        for (int i = 0; i < size; i++) {
            System.out.print(deque[(front + i) % capacity] + " ");
        }
        System.out.println();
    }
}
public class DequeDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the capacity of the Deque: ");
        int capacity = scanner.nextInt();
        Deque deque = new Deque(capacity);
        while (true) {
```

```
            System.out.println("\nMenu:");
            System.out.println("1. Insert at Front");
            System.out.println("2. Insert at Rear");
            System.out.println("3. Delete from Front");
            System.out.println("4. Delete from Rear");
            System.out.println("5. Display");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Enter value to insert at front: ");
                    int frontValue = scanner.nextInt();
                    deque.insertFront(frontValue);
                    break;
                case 2:
                    System.out.print("Enter value to insert at rear: ");
                    int rearValue = scanner.nextInt();
                    deque.insertRear(rearValue);
                    break;
                case 3:
                    int frontDeleted = deque.deleteFront();
                    if (frontDeleted != -1) {
                        System.out.println("Deleted from front: " + frontDeleted);
                    }
                    break;
                case 4:
                    int rearDeleted = deque.deleteRear();
                    if (rearDeleted != -1) {
                        System.out.println("Deleted from rear: " + rearDeleted);
                    }
                    break;
                case 5:
                    deque.display();
                    break;
                case 6:
                    System.out.println("Exiting...");
                    scanner.close();
                    System.exit(0);
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
}
```

Output:

```
PS D:\FYMCA\ADS> java DequeDemo
Enter the capacity of the Deque: 3

Menu:
1. Insert at Front
2. Insert at Rear
3. Delete from Front
4. Delete from Rear
5. Display
6. Exit
Enter your choice: 1
Enter value to insert at front: 10
10 inserted at the front.

Menu:
1. Insert at Front
2. Insert at Rear
3. Delete from Front
4. Delete from Rear
5. Display
6. Exit
Enter your choice: 2
Enter value to insert at rear: 20
20 inserted at the rear.

Menu:
1. Insert at Front
2. Insert at Rear
3. Delete from Front
4. Delete from Rear
5. Display
6. Exit
Enter your choice: 1
Enter value to insert at front: 30
30 inserted at the front.

Menu:
1. Insert at Front
2. Insert at Rear
3. Delete from Front
4. Delete from Rear
5. Display
6. Exit
Enter your choice: 3
Deleted from front: 30

Menu:
1. Insert at Front
2. Insert at Rear
3. Delete from Front
4. Delete from Rear
5. Display
6. Exit
Enter your choice: 5
Deque elements: 10 20
```

13) Write a program to implement Doubly Linked List.

Code:

```
import java.util.Scanner;
class Node {
    int data;
    Node next;
    Node prev;
    public Node(int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}
class DoublyLinkedList {
    private Node head;
    public DoublyLinkedList() {
        this.head = null;
    }
    public boolean isEmpty() {
        return head == null;
    }
    public void insertAtBeginning(int data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
            head = newNode;
        } else {
            newNode.next = head;
            head.prev = newNode;
            head = newNode;
        }
        System.out.println(data + " inserted at the beginning.");
    }
    public void insertAtEnd(int data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
            newNode.prev = temp;
        }
        System.out.println(data + " inserted at the end.");
    }
    public void deleteFromBeginning() {
```

```
    if (isEmpty()) {
        System.out.println("List is empty. Cannot delete from the beginning.");
        return;
    }
    int deletedData = head.data;
    if (head.next != null) {
        head = head.next;
        head.prev = null;
    } else {
        head = null;
    }
    System.out.println(deletedData + " deleted from the beginning.");
}
public void deleteFromEnd() {
    if (isEmpty()) {
        System.out.println("List is empty. Cannot delete from the end.");
        return;
    }
    if (head.next == null) {
        int deletedData = head.data;
        head = null;
        System.out.println(deletedData + " deleted from the end.");
    } else {
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        int deletedData = temp.data;
        temp.prev.next = null;
        System.out.println(deletedData + " deleted from the end.");
    }
}
public void displayFromBeginning() {
    if (isEmpty()) {
        System.out.println("List is empty.");
        return;
    }
    Node temp = head;
    System.out.print("List from beginning: ");
    while (temp != null) {
        System.out.print(temp.data + " ");
        temp = temp.next;
    }
    System.out.println();
}
public void displayFromEnd() {
    if (isEmpty()) {
        System.out.println("List is empty.");
        return;
```

```
        }
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        System.out.print("List from end: ");
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.prev;
        }
        System.out.println();
    }
}
public class DoublyLinkedListDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        DoublyLinkedList dll = new DoublyLinkedList();
        while (true) {
            System.out.println("\nMenu:");
            System.out.println("1. Insert at Beginning");
            System.out.println("2. Insert at End");
            System.out.println("3. Delete from Beginning");
            System.out.println("4. Delete from End");
            System.out.println("5. Display from Beginning");
            System.out.println("6. Display from End");
            System.out.println("7. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Enter value to insert at beginning: ");
                    int beginValue = scanner.nextInt();
                    dll.insertAtBeginning(beginValue);
                    break;
                case 2:
                    System.out.print("Enter value to insert at end: ");
                    int endValue = scanner.nextInt();
                    dll.insertAtEnd(endValue);
                    break;
                case 3:
                    dll.deleteFromBeginning();
                    break;
                case 4:
                    dll.deleteFromEnd();
                    break;
                case 5:
                    dll.displayFromBeginning();
                    break;
                case 6:
```

```
                dll.displayFromEnd();
                break;
            case 7:
                System.out.println("Exiting...");
                scanner.close();
                System.exit(0);
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
}
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\DoublyLinkedListDemo.java
PS D:\FYMCA\ADS> java DoublyLinkedListDemo

Menu:
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display from Beginning
6. Display from End
7. Exit
Enter your choice: 1
Enter value to insert at beginning: 10
10 inserted at the beginning.

Menu:
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display from Beginning
6. Display from End
7. Exit
Enter your choice: 2
Enter value to insert at end: 20
20 inserted at the end.

Menu:
1. Insert at Beginning
2. Insert at End
3. Delete from Beginning
4. Delete from End
5. Display from Beginning
6. Display from End
7. Exit
Enter your choice: 5
List from beginning: 10 20
```

14) Write a program to implement Adjacency matrix.

Code:

```
import java.util.Scanner;
class Graph {
   private int[][] adjacencyMatrix;
   private int numberOfVertices;
   public Graph(int numberOfVertices) {
      this.numberOfVertices = numberOfVertices;
      adjacencyMatrix = new int[numberOfVertices][numberOfVertices];
   }
   public void addEdge(int vertex1, int vertex2) {
      if (vertex1 < numberOfVertices && vertex2 < numberOfVertices) {
         adjacencyMatrix[vertex1][vertex2] = 1; // Directed graph (edge from vertex1 to
vertex2)
         adjacencyMatrix[vertex2][vertex1] = 1; // Since it's an undirected graph
         System.out.println("Edge added between vertex " + vertex1 + " and vertex " +
vertex2);
      } else {
         System.out.println("Invalid vertex numbers.");
      }
   public void removeEdge(int vertex1, int vertex2) {
      if (vertex1 < numberOfVertices && vertex2 < numberOfVertices) {
         adjacencyMatrix[vertex1][vertex2] = 0;
         adjacencyMatrix[vertex2][vertex1] = 0;
         System.out.println("Edge removed between vertex " + vertex1 + " and vertex " +
vertex2);
      } else {
         System.out.println("Invalid vertex numbers.");
      }
   }
   public void display() {
      System.out.println("\nAdjacency Matrix:");
      for (int i = 0; i < numberOfVertices; i++) {
         for (int j = 0; j < numberOfVertices; j++) {
            System.out.print(adjacencyMatrix[i][j] + " ");
         }
         System.out.println();
      }
   }
   public boolean isEdge(int vertex1, int vertex2) {
      if (vertex1 < numberOfVertices && vertex2 < numberOfVertices) {
         return adjacencyMatrix[vertex1][vertex2] == 1;
      } else {
         System.out.println("Invalid vertex numbers.");
         return false;
      }
```

```
    }
}
public class AdjacencyMatrixGraph {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of vertices: ");
        int vertices = scanner.nextInt();
        Graph graph = new Graph(vertices);
        while (true) {
            System.out.println("\nMenu:");
            System.out.println("1. Add an edge");
            System.out.println("2. Remove an edge");
            System.out.println("3. Display the adjacency matrix");
            System.out.println("4. Check if there is an edge");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Enter vertex 1: ");
                    int vertex1 = scanner.nextInt();
                    System.out.print("Enter vertex 2: ");
                    int vertex2 = scanner.nextInt();
                    graph.addEdge(vertex1, vertex2);
                    break;
                case 2:
                    System.out.print("Enter vertex 1: ");
                    int vertex1ToRemove = scanner.nextInt();
                    System.out.print("Enter vertex 2: ");
                    int vertex2ToRemove = scanner.nextInt();
                    graph.removeEdge(vertex1ToRemove, vertex2ToRemove);
                    break;
                case 3:
                    graph.display();
                    break;
                case 4:
                    System.out.print("Enter vertex 1: ");
                    int vertex1Check = scanner.nextInt();
                    System.out.print("Enter vertex 2: ");
                    int vertex2Check = scanner.nextInt();
                    boolean isEdge = graph.isEdge(vertex1Check, vertex2Check);
                    if (isEdge) {
                        System.out.println("There is an edge between vertex " + vertex1Check + "
and vertex " + vertex2Check);
                    } else {
                        System.out.println("No edge between vertex " + vertex1Check + " and
vertex " + vertex2Check);
                    }
                    break;
```

```
            case 5:
                System.out.println("Exiting...");
                scanner.close();
                System.exit(0);
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
  }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\AdjacencyMatrixGraph.java
PS D:\FYMCA\ADS> java AdjacencyMatrixGraph
Enter the number of vertices: 3

Menu:
1. Add an edge
2. Remove an edge
3. Display the adjacency matrix
4. Check if there is an edge
5. Exit
Enter your choice: 1
Enter vertex 1: 0
Enter vertex 2: 1
Edge added between vertex 0 and vertex 1

Menu:
1. Add an edge
2. Remove an edge
3. Display the adjacency matrix
4. Check if there is an edge
5. Exit
Enter your choice: 1
Enter vertex 1: 1
Enter vertex 2: 2
Edge added between vertex 1 and vertex 2

Menu:
1. Add an edge
2. Remove an edge
3. Display the adjacency matrix
4. Check if there is an edge
5. Exit
Enter your choice: 3

Adjacency Matrix:
0 1 0
1 0 1
0 1 0
```

15) Write a menu driven program to implement DFS and BFS.

Code:

```
import java.util.*;

class Graph {
    private int numberOfVertices;
    private List<LinkedList<Integer>> adjacencyList;
    public Graph(int numberOfVertices) {
        this.numberOfVertices = numberOfVertices;
        adjacencyList = new ArrayList<>();
        for (int i = 0; i < numberOfVertices; i++) {
            adjacencyList.add(new LinkedList<Integer>());
        }
    }
    public void addEdge(int vertex1, int vertex2) {
        adjacencyList.get(vertex1).add(vertex2); // Directed edge from vertex1 to vertex2
        adjacencyList.get(vertex2).add(vertex1); // Since it's an undirected graph
        System.out.println("Edge added between " + vertex1 + " and " + vertex2);
    }
    public void dfs(int startVertex) {
        boolean[] visited = new boolean[numberOfVertices];
        System.out.print("DFS Traversal starting from vertex " + startVertex + ": ");
        dfsUtil(startVertex, visited);
        System.out.println();
    }
    private void dfsUtil(int vertex, boolean[] visited) {
        visited[vertex] = true;
        System.out.print(vertex + " ");
        for (int adjacent : adjacencyList.get(vertex)) {
            if (!visited[adjacent]) {
                dfsUtil(adjacent, visited);
            }
        }
    }
    public void bfs(int startVertex) {
        boolean[] visited = new boolean[numberOfVertices];
        LinkedList<Integer> queue = new LinkedList<>(); // Using LinkedList as a queue
        visited[startVertex] = true;
        queue.add(startVertex);
        System.out.print("BFS Traversal starting from vertex " + startVertex + ": ");
        while (!queue.isEmpty()) {
            int vertex = queue.poll();
            System.out.print(vertex + " ");

            for (int adjacent : adjacencyList.get(vertex)) {
                if (!visited[adjacent]) {
                    visited[adjacent] = true;
```

```
                queue.add(adjacent);
            }
        }
    }
    System.out.println();
  }
  public void display() {
    System.out.println("\nAdjacency List:");
    for (int i = 0; i < numberOfVertices; i++) {
      System.out.print(i + ": ");
      for (int vertex : adjacencyList.get(i)) {
        System.out.print(vertex + " ");
      }
      System.out.println();
    }
  }
}
public class GraphTraversal {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the number of vertices: ");
    int vertices = scanner.nextInt();
    Graph graph = new Graph(vertices);
    while (true) {
      System.out.println("\nMenu:");
      System.out.println("1. Add an edge");
      System.out.println("2. Display the graph");
      System.out.println("3. Perform DFS");
      System.out.println("4. Perform BFS");
      System.out.println("5. Exit");
      System.out.print("Enter your choice: ");
      int choice = scanner.nextInt();
      switch (choice) {
        case 1:
          System.out.print("Enter vertex 1: ");
          int vertex1 = scanner.nextInt();
          System.out.print("Enter vertex 2: ");
          int vertex2 = scanner.nextInt();
          graph.addEdge(vertex1, vertex2);
          break;
        case 2:
          graph.display();
          break;
        case 3:
          System.out.print("Enter the starting vertex for DFS: ");
          int dfsStart = scanner.nextInt();
          graph.dfs(dfsStart);
          break;
        case 4:
```

```
                System.out.print("Enter the starting vertex for BFS: ");
                int bfsStart = scanner.nextInt();
                graph.bfs(bfsStart);
                break;
            case 5:
                System.out.println("Exiting...");
                scanner.close();
                System.exit(0);
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
  }
}
```

Output:

```
Enter the number of vertices: 5

Menu:
1. Add an edge
2. Display the graph
3. Perform DFS
4. Perform BFS
5. Exit
Enter your choice: 1
Enter vertex 1: 0
Enter vertex 2: 1
Edge added between 0 and 1

Menu:
Enter your choice: 1
Enter vertex 1: 1
Enter vertex 2: 2
Edge added between 1 and 2

Menu:
Enter your choice: 3
Enter the starting vertex for DFS: 0
DFS Traversal starting from vertex 0: 0 1 2
```

MRIGANKA ADHIKARY
ROLL NO: 69

16) Write a program to implement different operations on Binary Search Tree.

Code:

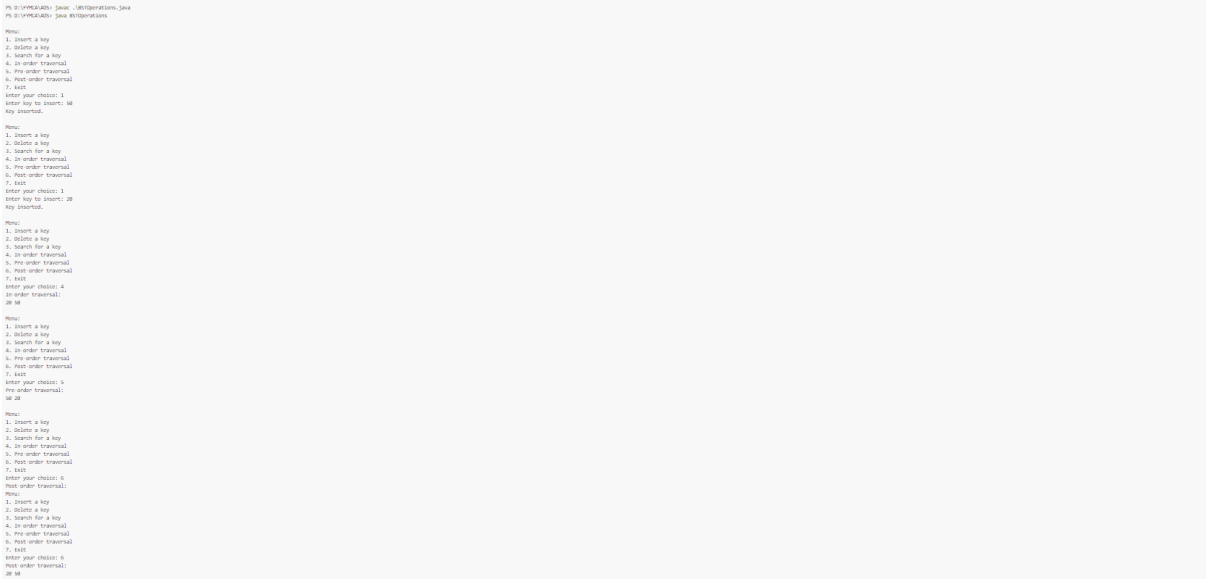
```
import java.util.Scanner;
class BinarySearchTree {
   static class Node {
      int key;
      Node left, right;

      public Node(int item) {
         key = item;
         left = right = null;
      }
   }
   Node root;
   BinarySearchTree() {
      root = null;
   }
   void insert(int key) {
      root = insertRec(root, key);
   }
   Node insertRec(Node root, int key) {
      if (root == null) {
         root = new Node(key);
         return root;
      }
      if (key < root.key)
         root.left = insertRec(root.left, key);
      else if (key > root.key)
         root.right = insertRec(root.right, key);
      return root;
   }
   void deleteKey(int key) {
      root = deleteRec(root, key);
   }
   Node deleteRec(Node root, int key) {
      if (root == null)
         return root;
      if (key < root.key)
         root.left = deleteRec(root.left, key);
      else if (key > root.key)
         root.right = deleteRec(root.right, key);
      else {
         if (root.left == null)
            return root.right;
         else if (root.right == null)
            return root.left;
```

```
        root.key = minValue(root.right);
        root.right = deleteRec(root.right, root.key);
    }
    return root;
}
int minValue(Node root) {
    int minValue = root.key;
    while (root.left != null) {
        minValue = root.left.key;
        root = root.left;
    }
    return minValue;
}
boolean search(int key) {
    return searchRec(root, key);
}
boolean searchRec(Node root, int key) {
    if (root == null)
        return false;
    if (root.key == key)
        return true;
    return key < root.key ? searchRec(root.left, key) : searchRec(root.right, key);
}
void inorder() {
    inorderRec(root);
    System.out.println();
}
void inorderRec(Node root) {
    if (root != null) {
        inorderRec(root.left);
        System.out.print(root.key + " ");
        inorderRec(root.right);
    }
}
void preorder() {
    preorderRec(root);
    System.out.println();
}
void preorderRec(Node root) {
    if (root != null) {
        System.out.print(root.key + " ");
        preorderRec(root.left);
        preorderRec(root.right);
    }
}
void postorder() {
    postorderRec(root);
    System.out.println();
}
```

```
    void postorderRec(Node root) {
        if (root != null) {
            postorderRec(root.left);
            postorderRec(root.right);
            System.out.print(root.key + " ");
        }
    }
}
public class BSTOperations {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        BinarySearchTree bst = new BinarySearchTree();
        while (true) {
            System.out.println("\nMenu:");
            System.out.println("1. Insert a key");
            System.out.println("2. Delete a key");
            System.out.println("3. Search for a key");
            System.out.println("4. In-order traversal");
            System.out.println("5. Pre-order traversal");
            System.out.println("6. Post-order traversal");
            System.out.println("7. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Enter key to insert: ");
                    int keyToInsert = scanner.nextInt();
                    bst.insert(keyToInsert);
                    System.out.println("Key inserted.");
                    break;
                case 2:
                    System.out.print("Enter key to delete: ");
                    int keyToDelete = scanner.nextInt();
                    bst.deleteKey(keyToDelete);
                    System.out.println("Key deleted (if it existed).");
                    break;
                case 3:
                    System.out.print("Enter key to search: ");
                    int keyToSearch = scanner.nextInt();
                    if (bst.search(keyToSearch))
                        System.out.println("Key found.");
                    else
                        System.out.println("Key not found.");
                    break;
                case 4:
                    System.out.println("In-order traversal:");
                    bst.inorder();
                    break;
                case 5:
```

```
                System.out.println("Pre-order traversal:");
                bst.preorder();
                break;
            case 6:
                System.out.println("Post-order traversal:");
                bst.postorder();
                break;
            case 7:
                System.out.println("Exiting...");
                scanner.close();
                System.exit(0);
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
  }
}
```

Output:



17) Write a menu driven program for implementing various Hashing techniques with Linear Probing.

Code:

```
import java.util.Scanner;

class HashTable {
    private int[] table;
    private int size;
    private static final int DELETED = Integer.MIN_VALUE;
    public HashTable(int size) {
        this.size = size;
        table = new int[size];
        for (int i = 0; i < size; i++) {
            table[i] = -1; // Initialize table with -1 (empty slots)
        }
    }
    private int hashFunction(int key) {
        return key % size;
    }
    public void insert(int key) {
        int index = hashFunction(key);
        int originalIndex = index;
        boolean inserted = false;
        do {
            if (table[index] == -1 || table[index] == DELETED) { // Empty or deleted slot
                table[index] = key;
                inserted = true;
                System.out.println("Key " + key + " inserted at index " + index);
                break;
            }
            index = (index + 1) % size; // Linear probing
        } while (index != originalIndex);
        if (!inserted) {
            System.out.println("Hash table is full. Cannot insert key " + key);
        }
    }
    public boolean search(int key) {
        int index = hashFunction(key);
        int originalIndex = index;
        do {
            if (table[index] == key) {
                System.out.println("Key " + key + " found at index " + index);
                return true;
            } else if (table[index] == -1) {
                break; // Stop if an empty slot is found
            }
```

```
            index = (index + 1) % size; // Linear probing
        } while (index != originalIndex);
        System.out.println("Key " + key + " not found.");
        return false;
    }
    public void delete(int key) {
        int index = hashFunction(key);
        int originalIndex = index;
        do {
            if (table[index] == key) {
                table[index] = DELETED; // Mark the slot as deleted
                System.out.println("Key " + key + " deleted from index " + index);
                return;
            } else if (table[index] == -1) {
                break; // Stop if an empty slot is found
            }
            index = (index + 1) % size; // Linear probing
        } while (index != originalIndex);

        System.out.println("Key " + key + " not found. Cannot delete.");
    }
    public void display() {
        System.out.println("\nHash Table:");
        for (int i = 0; i < size; i++) {
            if (table[i] == -1) {
                System.out.println(i + ": EMPTY");
            } else if (table[i] == DELETED) {
                System.out.println(i + ": DELETED");
            } else {
                System.out.println(i + ": " + table[i]);
            }
        }
    }
}
public class HashingWithLinearProbing {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the hash table: ");
        int size = scanner.nextInt();
        HashTable hashTable = new HashTable(size);
        while (true) {
            System.out.println("\nMenu:");
            System.out.println("1. Insert a key");
            System.out.println("2. Search for a key");
            System.out.println("3. Delete a key");
            System.out.println("4. Display the hash table");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
```

```
            switch (choice) {
                case 1:
                    System.out.print("Enter key to insert: ");
                    int keyToInsert = scanner.nextInt();
                    hashTable.insert(keyToInsert);
                    break;
                case 2:
                    System.out.print("Enter key to search: ");
                    int keyToSearch = scanner.nextInt();
                    hashTable.search(keyToSearch);
                    break;
                case 3:
                    System.out.print("Enter key to delete: ");
                    int keyToDelete = scanner.nextInt();
                    hashTable.delete(keyToDelete);
                    break;
                case 4:
                    hashTable.display();
                    break;
                case 5:
                    System.out.println("Exiting...");
                    scanner.close();
                    System.exit(0);
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        }
    }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\HashingWithLinearProbing.java
PS D:\FYMCA\ADS> java HashingWithLinearProbing
Enter the size of the hash table: 3

Menu:
1. Insert a key
2. Search for a key
3. Delete a key
4. Display the hash table
5. Exit
Enter your choice: 1
Enter key to insert: 1
Key 1 inserted at index 1

Menu:
1. Insert a key
2. Search for a key
3. Delete a key
4. Display the hash table
5. Exit
Enter your choice: 1
Enter key to insert: 2
Key 2 inserted at index 2

Menu:
1. Insert a key
2. Search for a key
3. Delete a key
4. Display the hash table
5. Exit
Enter your choice: 4

Hash Table:
0: EMPTY
1: 1
2: 2
```

18) Write a program to implement Heap with different operations (Create, Insert, Delete).

Code:

```
import java.util.Scanner;
class Heap {
    private int[] heap;
    private int size;
    private int capacity;
    public Heap(int capacity) {
        this.capacity = capacity;
        this.size = 0;
        this.heap = new int[capacity];
    }
    private int parent(int i) {
        return (i - 1) / 2;
    }
    private int leftChild(int i) {
        return 2 * i + 1;
    }
    private int rightChild(int i) {
        return 2 * i + 2;
    }
    private void swap(int i, int j) {
        int temp = heap[i];
        heap[i] = heap[j];
        heap[j] = temp;
    }
    public void insert(int value) {
        if (size == capacity) {
            System.out.println("Heap is full. Cannot insert.");
            return;
        }
        heap[size] = value;
        size++;
        int current = size - 1;
        while (current != 0 && heap[current] > heap[parent(current)]) {
            swap(current, parent(current));
            current = parent(current);
        }
        System.out.println("Inserted " + value + " into the heap.");
    }
    public int delete() {
        if (size == 0) {
            System.out.println("Heap is empty. Cannot delete.");
            return -1;
        }
        int root = heap[0];
```

```
        heap[0] = heap[size - 1];
        size--;
        heapify(0);
        System.out.println("Deleted root: " + root);
        return root;
    }
    private void heapify(int i) {
        int largest = i;
        int left = leftChild(i);
        int right = rightChild(i);
        if (left < size && heap[left] > heap[largest]) {
            largest = left;
        }
        if (right < size && heap[right] > heap[largest]) {
            largest = right;
        }
        if (largest != i) {
            swap(i, largest);
            heapify(largest);
        }
    }
    public void display() {
        if (size == 0) {
            System.out.println("Heap is empty.");
            return;
        }
        System.out.print("Heap elements: ");
        for (int i = 0; i < size; i++) {
            System.out.print(heap[i] + " ");
        }
        System.out.println();
    }
}
public class HeapOperations {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the capacity of the heap: ");
        int capacity = scanner.nextInt();
        Heap heap = new Heap(capacity);
        while (true) {
            System.out.println("\nMenu:");
            System.out.println("1. Insert a value");
            System.out.println("2. Delete the root");
            System.out.println("3. Display the heap");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
```

```
                System.out.print("Enter the value to insert: ");
                int value = scanner.nextInt();
                heap.insert(value);
                break;
            case 2:
                heap.delete();
                break;
            case 3:
                heap.display();
                break;
            case 4:
                System.out.println("Exiting...");
                scanner.close();
                System.exit(0);
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
}
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\HeapOperations.java
PS D:\FYMCA\ADS> java HeapOperations
Enter the capacity of the heap: 3

Menu:
1. Insert a value
2. Delete the root
3. Display the heap
4. Exit
Enter your choice: 1
Enter the value to insert: 10
Inserted 10 into the heap.

Menu:
1. Insert a value
2. Delete the root
3. Display the heap
4. Exit
Enter your choice: 1
Enter the value to insert: 20
Inserted 20 into the heap.

Menu:
1. Insert a value
2. Delete the root
3. Display the heap
4. Exit
Enter your choice: 2
Deleted root: 20

Menu:
1. Insert a value
2. Delete the root
3. Display the heap
4. Exit
Enter your choice: 3
Heap elements: 10
```

19) Write a program to implement Minimum Spanning Tree using either Kruskal's or Prim's Algorithm.

Code:

```
import java.util.*;
class Edge implements Comparable<Edge> {
    int source, destination, weight;
    public Edge(int source, int destination, int weight) {
        this.source = source;
        this.destination = destination;
        this.weight = weight;
    }
    @Override
    public int compareTo(Edge other) {
        return this.weight - other.weight;
    }
}
class DisjointSet {
    private int[] parent, rank;
    public DisjointSet(int vertices) {
        parent = new int[vertices];
        rank = new int[vertices];
        for (int i = 0; i < vertices; i++) {
            parent[i] = i;
            rank[i] = 0;
        }
    }
    public int find(int vertex) {
        if (parent[vertex] != vertex) {
            parent[vertex] = find(parent[vertex]); // Path compression
        }
        return parent[vertex];
    }
    public void union(int root1, int root2) {
        if (rank[root1] > rank[root2]) {
            parent[root2] = root1;
        } else if (rank[root1] < rank[root2]) {
            parent[root1] = root2;
        } else {
            parent[root2] = root1;
            rank[root1]++;
        }
    }
}
public class KruskalsAlgorithm {
    private int vertices;
    private List<Edge> edges;
    public KruskalsAlgorithm(int vertices) {
```

```
        this.vertices = vertices;
        edges = new ArrayList<>();
    }
    public void addEdge(int source, int destination, int weight) {
        edges.add(new Edge(source, destination, weight));
    }
    public void findMST() {
        Collections.sort(edges);
        DisjointSet disjointSet = new DisjointSet(vertices);
        List<Edge> mst = new ArrayList<>();
        int mstWeight = 0;
        for (Edge edge : edges) {
            int root1 = disjointSet.find(edge.source);
            int root2 = disjointSet.find(edge.destination);
            if (root1 != root2) {
                mst.add(edge);
                mstWeight += edge.weight;
                disjointSet.union(root1, root2);
            }
            if (mst.size() == vertices - 1) {
                break;
            }
        }
        System.out.println("Edges in the Minimum Spanning Tree:");
        for (Edge edge : mst) {
            System.out.println(edge.source + " -- " + edge.destination + " == " +
edge.weight);
        }
        System.out.println("Total weight of MST: " + mstWeight);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of vertices: ");
        int vertices = scanner.nextInt();
        KruskalsAlgorithm graph = new KruskalsAlgorithm(vertices);
        System.out.print("Enter the number of edges: ");
        int edges = scanner.nextInt();
        System.out.println("Enter the edges (source destination weight):");
        for (int i = 0; i < edges; i++) {
            int source = scanner.nextInt();
            int destination = scanner.nextInt();
            int weight = scanner.nextInt();
            graph.addEdge(source, destination, weight);
        }
        graph.findMST();
        scanner.close();
    }
}
```

Output:

```
PS D:\FYMCA\ADS> javac .\KruskalsAlgorithm.java
PS D:\FYMCA\ADS> java KruskalsAlgorithm
Enter the number of vertices: 4
Enter the number of edges: 5
Enter the edges (source destination weight):
0
1
10
0
2
6
0
3
5
1
3
14

3
4
2
Invalid edge: (3, 4). Skipping...
Edges in the Minimum Spanning Tree:
0 -- 3 == 5
0 -- 2 == 6
0 -- 1 == 10
Total weight of MST: 21
```