Index				
Sr. No.	**Title**	**CO**	**Date**	**Sign**
1.	**Implementation of different sorting techniques**	**CO1**	**9/10/2024**	
	a) To implement bubble sort			
	b) To implement insertion sort			
	c) To implement selection sort			
	d) To implement shell sort			
	e) To implement radix sort			
	f) To implement quick sort			
2.	**Implementation of different searching techniques**	**CO1**	**9/10/2024**	
	a) To implement linear search			
	b) To implement binary search			
3.	**Implementation of Stacks (Using Arrays)**	**CO2**	**16/10/2024**	
	a) To implement Stack using array			
	b) To implement Stack using linked list			
4.	**Implementation of Stack Application**	**CO2**	**18/10/2024**	
	a) To implement Postfix evaluation			
	b) To implement Balancing of Parenthesis			
5.	**Implement all different types of queues**	**CO2**	**23/10/2024**	
	a) To implement Circular Queue			
6.	**Demonstrate application of queue**	**CO2**	**25/10/2024**	
	a) To implement Priority Queue			

7.	**Implementation of all types of linked list**	**CO2**	**06/11/2024**	
	a) To implement Single Linked List			
	b) To implement Double Linked list			
	c) To implement Circular Linked List			
8.	**Demonstrate application of linked list**	**CO2**	**08/11/2024**	
	a) To implement Polynomial Addition			
	b) To implement Sparse Matrix			
9.	**Create and perform various operations on BST**	**CO3**	**15/11/2024**	
	a) Inserting node in BST			
	b) Deleting the node from BST			
	c) To find height of Tree			
	d) To perform Inorder			
	e) To perform Preorder			
	f) To perform Postorder			
	g) To find Maximum value of tree			
10.	**Implementing Heap with different operations performed**	**CO3**	**22/11/2024**	
	a) To perform insertion operation			
	b) To create Heap using Heapify method			
	c) To perform Heap sort			
	d) To delete the value in heap			
11.	**Create a graph storage structure**	**CO3**	**27/11/2024**	
	a) Adjacency Matrix			

12.	**Perform various hashing techniques with Linear Probe as collision resolution scheme.**	**CO4**	**29/11/2024**	
13.	**Create a minimum spanning tree using any method Kruskal's algorithm or Prim's algorithm**	**CO3**	**04/12/2024**	
14.	**Implementation of Graph Traversal**	**CO3**	**13/12/2024**	
	a) Implement Depth First Search (DFS)			
	b) Implement Breath First Search (BFS)			

Pratical No :1 . Implementation of different sorting techniques

a) **To implement bubble sort**

CODE :

```
import java.util.Scanner;

public class Main {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter size for an array: ");
      int n = sc.nextInt();
      int[] a = new int[n];

      System.out.println("Enter the elements of the array:");
      for (int i = 0; i < n; i++) {
         a[i] = sc.nextInt();
      }

      // Bubble Sort implementation
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n - 1; j++) {
            if (a[j] > a[j + 1]) {
               // Swap a[j] and a[j+1]
               int temp = a[j];
               a[j] = a[j + 1];
               a[j + 1] = temp;
            }
         }

         // Print the array after each pass
         for (int k = 0; k < n; k++) {
            System.out.print(a[k] + " ");
         }
         System.out.println();
      }

      System.out.println("Sorted array:");
      for (int i = 0; i < n; i++) {
```

```
            System.out.print(a[i] + " ");
        }

        sc.close();
    }
}
```

OUTPUT :

```
enter size for an array: 5
21 10 21 52 35
 10 21 21 35 52
 10 21 21 35 52
 10 21 21 35 52
 10 21 21 35 52
 10 21 21 35 52
sorted array
 10 21 21 35 52
--------------------------------
Process exited after 361.5 seconds with return value 0
Press any key to continue . . .
```

b) To implement Insertion Sort :

CODE :

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter size for an array: ");
        int n = sc.nextInt();
        int[] a = new int[n];

        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            a[i] = sc.nextInt();
        }

        // Insertion Sort
        for (int i = 0; i < n - 1; i++) {
```

```
            for (int j = i + 1; j > 0; j--) {
                if (a[j] < a[j - 1]) {
                    // Swap a[j] and a[j-1]
                    int temp = a[j];
                    a[j] = a[j - 1];
                    a[j - 1] = temp;
                }
            }

            // Print the array after each pass
            for (int k = 0; k < n; k++) {
                System.out.print(a[k] + " ");
            }
            System.out.println();
        }

        System.out.println("Sorted array:");
        for (int i = 0; i < n; i++) {
            System.out.print(a[i] + " ");
        }

        sc.close();
    }
}
```

OUTPUT :

```
enter size for an array: 5
212 231 654 257 625
 212 231 654 257 625
 212 231 654 257 625
 212 231 257 654 625
 212 231 257 625 654
selection sort is:  212 231 257 625 654
--------------------------------
Process exited after 24.05 seconds with return value 0
Press any key to continue . . .
```

c) To implement Selection Sort :

CODE :

```
import java.util.Scanner;

public class SelectionSortSteps {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input the size of the array
        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();

        // Input array elements
        int[] arr = new int[n];
        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.println("Original array:");
        printArray(arr);

        // Perform Selection Sort with steps
        selectionSortWithSteps(arr);

        System.out.println("Sorted array:");
        printArray(arr);

        scanner.close();
    }

    // Selection sort with step-by-step output
    public static void selectionSortWithSteps(int[] arr) {
        int n = arr.length;

        for (int i = 0; i < n - 1; i++) {
            int minIndex = i; // Index of the smallest element
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
```

```
                minIndex = j;
            }
        }
        // Swap the smallest element with the first element of
the unsorted portion
        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;

        // Print the array after each pass
        System.out.println("\nAfter pass " + (i + 1) + ":");
        printArray(arr);
      }
    }

    // Utility method to print the array
    public static void printArray(int[] arr) {
      for (int num : arr) {
        System.out.print(num + " ");
      }
      System.out.println();
    }
}
```

OUTPUT :

```
C:\Users\Atharva Mahulkar\OneDrive\Desktop\Ds_codes>java SelectionSortSteps
Enter the number of elements: 5
Enter 5 elements:
32
78
14
90
66
Original array:
32 78 14 90 66

After pass 1:
14 78 32 90 66

After pass 2:
14 32 78 90 66

After pass 3:
14 32 66 90 78

After pass 4:
14 32 66 78 90
Sorted array:
14 32 66 78 90
```

c) To implement Shell Sort :

CODE :

```
import java.util.Scanner;

public class ShellSortSteps {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input the size of the array
        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();

        // Input array elements
        int[] arr = new int[n];
        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        System.out.println("Original array:");
        printArray(arr);

        // Perform Shell Sort with steps
        shellSortWithSteps(arr);

        System.out.println("Sorted array:");
        printArray(arr);

        scanner.close();
    }

    // Shell Sort with step-by-step output
    public static void shellSortWithSteps(int[] arr) {
        int n = arr.length;

        // Start with a large gap, then reduce the gap
        for (int gap = n / 2; gap > 0; gap /= 2) {
            System.out.println("\nGap = " + gap + ":");
```

```
            // Perform a gapped insertion sort
            for (int i = gap; i < n; i++) {
                int temp = arr[i];
                int j;

                // Shift earlier gap-sorted elements up until the
   correct location is found
                for (j = i; j >= gap && arr[j - gap] > temp; j -=
   gap) {
                    arr[j] = arr[j - gap];
                }

                // Put temp (the original arr[i]) in its correct
   location
                arr[j] = temp;

                // Print the array after each insertion
                printArray(arr);
            }
        }
    }

    // Utility method to print the array
    public static void printArray(int[] arr) {
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

OUTPUT :

```
C:\Users\Atharva Mahulkar\OneDrive\Desktop\Ds_codes>java ShellSortSteps
Enter the number of elements: 5
Enter 5 elements:
12
90
34
12
87
Original array:
12 90 34 12 87

Gap = 2:
12 90 34 12 87
12 12 34 90 87
12 12 34 90 87

Gap = 1:
12 12 34 90 87
12 12 34 90 87
12 12 34 90 87
12 12 34 87 90
Sorted array:
12 12 34 87 90
```

a) To implement radix sort

CODE :

```
import java.util.Arrays;

public class RadixSort {

    // Main function to implement Radix Sort

    public static void radixSort(int[] arr) {

        // Find the maximum number to determine the number of digits

        int max = Arrays.stream(arr).max().getAsInt();

        // Apply counting sort for each digit

        for (int exp = 1; max / exp > 0; exp *= 10) {

            countingSort(arr, exp);
```

```
    }
  }
  // Counting sort used as a subroutine
  private static void countingSort(int[] arr, int exp) {
    int n = arr.length;
    int[] output = new int[n]; // Output array to store sorted numbers
    int[] count = new int[10]; // Count array for digits 0-9

    // Count the occurrences of each digit at the current position
    for (int i = 0; i < n; i++) {
      int digit = (arr[i] / exp) % 10;
      count[digit]++;
    }
    // Update count[i] to store the actual position of this digit in output
    for (int i = 1; i < 10; i++) {
      count[i] += count[i - 1];
    }
```

```
        // Build the output array by placing numbers in sorted order of the current digit
        for (int i = n - 1; i >= 0; i--) {
            int digit = (arr[i] / exp) % 10;
            output[count[digit] - 1] = arr[i];
            count[digit]--;
        }
        // Copy the sorted numbers back to the original array
        System.arraycopy(output, 0, arr, 0, n);
    }
    // Driver function to test the algorithm
    public static void main(String[] args) {
        int[] arr = {170, 45, 75, 90, 802, 24, 2, 66};
        System.out.println("Original Array: " + Arrays.toString(arr));
        radixSort(arr);
      System.out.println("Sorted Array: " + Arrays.toString(arr));
    }
}
```

OUTPUT :

```
C:\Users\Atharva Mahulkar\OneDrive\Desktop\Ds_codes>java RadixSort
Original Array: [170, 45, 75, 90, 802, 24, 2, 66]
Sorted Array: [2, 24, 45, 66, 75, 90, 170, 802]
```

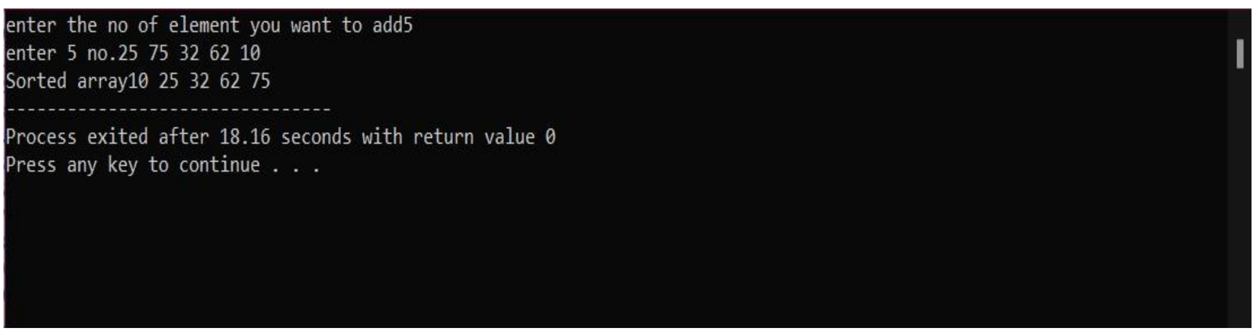
a) To implement Quick sort
CODE :

```
import java.util.Arrays;

public class QuickSort {
   // Function to perform
Quick Sort
   public static void
quickSort(int[] arr, int low,
int high) {
      if (low < high) {
         // Partition the array
and get the pivot index
         int pi = partition(arr,
low, high);

         // Recursively sort
elements before and after the
pivot
         quickSort(arr, low, pi
- 1);
         quickSort(arr, pi + 1,
high);
      }
   }

   // Partition function
```

```
    private static int
partition(int[] arr, int low,
int high) {
        int pivot = arr[high]; //
Pivot element
        int i = low - 1; // Index
of smaller element
        for (int j = low; j < high;
j++) {
            // If the current
element is smaller than the
pivot
            if (arr[j] < pivot) {
                i++;
                // Swap arr[i] and
arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // Swap arr[i+1] and
arr[high] (the pivot)
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1; // Return
the pivot index
    }

    // Main method to test the
Quick Sort
    public static void
main(String[] args) {
        int[] arr = {10, 80, 30,
90, 40, 50, 70};
```

```
System.out.println("Origina
l Array: " +
Arrays.toString(arr));

      quickSort(arr, 0,
arr.length - 1);


System.out.println("Sorted
Array: " +
Arrays.toString(arr));
   }
}
```

OUTPUT :



Pratical No : 2. Implementation to different searching techniques

a) **To implement Linear search**

CODE :

```
public class LinearSearch {
   // Function to perform Linear
Search
   public static int
linearSearch(int[] arr, int
target) {
      for (int i = 0; i < arr.length;
i++) {
```

```
            if (arr[i] == target) {
                return i; // Return the
index if found
            }
        }
        return -1; // Return -1 if
not found
    }

    public static void
main(String[] args) {
        int[] arr = {10, 20, 30, 40,
50};
        int target = 30;

        int result =
linearSearch(arr, target);
        if (result != -1) {

System.out.println("Element
found at index: " + result);
        } else {

System.out.println("Element
not found.");
        }
    }
}
```

OUTPUT :

```
C:\Users\Atharva Mahulkar\OneDrive\Desktop\Ds_codes>java LinearSearch
Element found at index: 2

C:\Users\Atharva Mahulkar\OneDrive\Desktop\Ds_codes>
```

bTo implement Binary search

CODE :

```
import java.util.Arrays;

public class BinarySearch {
   // Function to perform Binary
Search
   public static int
binarySearch(int[] arr, int target) {
       int low = 0, high = arr.length -
1;
       while (low <= high) {
          int mid = low + (high - low)
/ 2; // Avoids overflow for large low
and high values
          if (arr[mid] == target) {
             return mid; // Return the
index if found
          } else if (arr[mid] < target)
{
             low = mid + 1; // Search
in the right subarray
          } else {
             high = mid - 1; // Search
in the left subarray
          }
       }
       return -1; // Return -1 if not
found
   }

   public static void main(String[]
args) {
       int[] arr = {10, 20, 30, 40, 50};
       int target = 30;

       int result = binarySearch(arr,
target);
       if (result != -1) {
```

```
System.out.println("Element
found at index: " + result);
      } else {

System.out.println("Element not
found.");
      }
   }
}
```

OUTPUT:

```
C:\Users\Atharva Mahulkar\OneDrive\Desktop\Ds_codes>java BinarySearch
Element found at index: 2

C:\Users\Atharva Mahulkar\OneDrive\Desktop\Ds_codes>
```

Pratical No : 3. Implementation of stacks (Using arrays and Linked List)

a) To implement stack using array

CODE :

```
import java.util.Scanner;

// Node class to represent each
element in the linked list
class Node {
   int data;
   Node next;

   // Constructor
   Node(int data) {
      this.data = data;
      this.next = null;
   }
}

// Stack class that implements stack
operations using a linked list
```

```
class Stack {
    private Node top;
    private int size;

    // Constructor
    public Stack() {
        this.top = null;
        this.size = 0;
    }

    // Push operation
    public void push(int data) {
        Node newNode = new
Node(data);
        newNode.next = top; // Point new
node to current top
        top = newNode; // Make new
node the top
        size++;
        System.out.println(data + "
pushed to stack");
    }

    // Pop operation
    public void pop() {
        if (isEmpty()) {
            System.out.println("Stack is
empty. Nothing to pop.");
            return;
        }
        int popped = top.data;
        top = top.next; // Remove top
node
        size--;
        System.out.println(popped + "
popped from stack");
    }

    // Peek operation
    public void peek() {
        if (isEmpty()) {
```

```
            System.out.println("Stack is
empty. Nothing to peek.");
            return;
        }
        System.out.println("Top element
is: " + top.data);
    }

    // Check if stack is empty
    public boolean isEmpty() {
        return top == null;
    }

    // Get size of stack
    public void getSize() {
        System.out.println("Size of
stack: " + size);
    }

    // Display all elements of the stack
    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is
empty.");
            return;
        }
        Node temp = top;
        System.out.print("Stack
elements: ");
        while (temp != null) {
            System.out.print(temp.data +
" ");
            temp = temp.next;
        }
        System.out.println();
    }
}

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

```
        Scanner scanner = new
Scanner(System.in);
        Stack stack = new Stack();
        int choice;

        // Menu-driven program
        do {
            System.out.println("\nStack
Operations:");
            System.out.println("1. Push");
            System.out.println("2. Pop");
            System.out.println("3. Peek");
            System.out.println("4. Check
if Stack is Empty");
            System.out.println("5. Get
Stack Size");
            System.out.println("6. Display
Stack");
            System.out.println("7. Exit");
            System.out.print("Enter your
choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    // Push operation
                    System.out.print("Enter
element to push: ");
                    int data =
scanner.nextInt();
                    stack.push(data);
                    break;
                case 2:
                    // Pop operation
                    stack.pop();
                    break;
                case 3:
                    // Peek operation
                    stack.peek();
                    break;
                case 4:
```

```
                // Check if stack is empty
                if (stack.isEmpty()) {
System.out.println("Stack is empty.");
                } else {
System.out.println("Stack is not empty.");
                }
                break;
            case 5:
                // Get size of stack
                stack.getSize();
                break;
            case 6:
                // Display stack
                stack.display();
                break;
            case 7:
System.out.println("Exiting...");
                break;
            default:
System.out.println("Invalid choice. Please
try again.");
        }
    } while (choice != 7);

    scanner.close();
  }
 }
}
```

OUTPUT:

```
C:\Users\Atharva Mahulkar\OneDrive\Desktop\Ds_codes>java StackUsingLinkedList

Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Stack is Empty
5. Get Stack Size
6. Display Stack
7. Exit
Enter your choice: 1
Enter element to push: 11
11 pushed to stack

Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Stack is Empty
5. Get Stack Size
6. Display Stack
7. Exit
Enter your choice: 1
Enter element to push: 22
22 pushed to stack

Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Stack is Empty
5. Get Stack Size
```

```
Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Stack is Empty
5. Get Stack Size
6. Display Stack
7. Exit
Enter your choice: 2
22 popped from stack

Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Stack is Empty
5. Get Stack Size
6. Display Stack
7. Exit
Enter your choice: 3
Top element is: 11

Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Stack is Empty
5. Get Stack Size
6. Display Stack
7. Exit
Enter your choice: 4
Stack is not empty.
```

a)To implement stack using linked list

CODE :

```
import java.util.Scanner;

// Stack class using an array
class Stack {
    private int maxSize;
    private int top;
    private int[] stack;

    // Constructor to initialize the stack with a maximum size
    public Stack(int size) {
        maxSize = size;
        stack = new int[maxSize];
        top = -1; // Stack is empty when top is -1
    }

    // Push operation
    public void push(int data) {
        if (top == maxSize - 1) {
            System.out.println("Stack Overflow. Cannot push "
+ data);
        } else {
            stack[++top] = data;
            System.out.println(data + " pushed to stack");
        }
    }

    // Pop operation
    public void pop() {
        if (isEmpty()) {
            System.out.println("Stack Underflow. Nothing to
pop.");
        } else {
            int popped = stack[top--];
            System.out.println(popped + " popped from stack");
        }
    }
```

```
    // Peek operation
    public void peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty. Nothing to
peek.");
        } else {
            System.out.println("Top element is: " + stack[top]);
        }
    }

    // Check if stack is empty
    public boolean isEmpty() {
        return top == -1;
    }

    // Get the size of the stack
    public void getSize() {
        System.out.println("Size of stack: " + (top + 1));
    }

    // Display all elements of the stack
    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
        } else {
            System.out.print("Stack elements: ");
            for (int i = 0; i <= top; i++) {
                System.out.print(stack[i] + " ");
            }
            System.out.println();
        }
    }
}

public class StackUsingArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        // Get the stack size from the user
        System.out.print("Enter the maximum size of the stack: 
");
        int size = scanner.nextInt();
        Stack stack = new Stack(size);

        int choice;

        // Menu-driven program
        do {
            System.out.println("\nStack Operations:");
            System.out.println("1. Push");
            System.out.println("2. Pop");
            System.out.println("3. Peek");
            System.out.println("4. Check if Stack is Empty");
            System.out.println("5. Get Stack Size");
            System.out.println("6. Display Stack");
            System.out.println("7. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    // Push operation
                    System.out.print("Enter element to push: ");
                    int data = scanner.nextInt();
                    stack.push(data);
                    break;
                case 2:
                    // Pop operation
                    stack.pop();
                    break;
                case 3:
                    // Peek operation
                    stack.peek();
                    break;
                case 4:
```

```
                // Check if stack is empty
                if (stack.isEmpty()) {
                    System.out.println("Stack is empty.");
                } else {
                    System.out.println("Stack is not empty.");
                }
                break;
            case 5:
                // Get size of stack
                stack.getSize();
                break;
            case 6:
                // Display stack
                stack.display();
                break;
            case 7:
                System.out.println("Exiting...");
                break;
            default:
                System.out.println("Invalid choice. Please try
again.");
          }
        } while (choice != 7);

        scanner.close();
    }
}
```

OUTPUT:

```
Stack Operations Using Linked List !!!

1) Push in stack
2) Pop from stack
3) Traverse stack
4) Exit
Enter choice:
1
Enter value to be pushed:
23
Enter choice:
1
Enter value to be pushed:
32
Enter choice:
1
Enter value to be pushed:
23
Enter choice:
1
Enter value to be pushed:
65
Enter choice:
1
Enter value to be pushed:
52
Enter choice:
2
The popped element is 52
Enter choice:
```

```
Enter value to be pushed:
32
Enter choice:
1
Enter value to be pushed:
23
Enter choice:
1
Enter value to be pushed:
65
Enter choice:
1
Enter value to be pushed:
52
Enter choice:
2
The popped element is 52
Enter choice:
3
Stack elements are: 65 23 32 23
Enter choice:
4
Exit

--------------------------------
Process exited after 68.21 seconds with return value 0
Press any key to continue . . .
```

Pratical No : 3. Implementation of stack Applicatiion

a) **To implement Postfix evaluation**
CODE :

```
import java.util.Scanner;
import java.util.Stack;

public class PostfixEvaluation {

   // Method to evaluate a postfix expression
   public static int evaluatePostfix(String expression) {
      Stack<Integer> stack = new Stack<>();

      // Traverse the expression
      for (int i = 0; i < expression.length(); i++) {
         char ch = expression.charAt(i);

         // If the character is a digit, push it to the stack
         if (Character.isDigit(ch)) {
            stack.push(ch - '0'); // Convert character to integer
         }
         // If the character is an operator, pop two elements,
apply the operator,
         // and push the result
         else {
            int operand2 = stack.pop(); // Second operand
            int operand1 = stack.pop(); // First operand

            switch (ch) {
               case '+':
                  stack.push(operand1 + operand2);
                  break;
               case '-':
                  stack.push(operand1 - operand2);
                  break;
               case '*':
                  stack.push(operand1 * operand2);
```

```
                break;
            case '/':
                stack.push(operand1 / operand2);
                break;
            default:
                throw new
IllegalArgumentException("Invalid operator: " + ch);
            }
        }
    }

    // The result is the only element left in the stack
    return stack.pop();
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.println("Enter a postfix expression (e.g.,
235+4*): ");
    String expression = scanner.nextLine();

    try {
        int result = evaluatePostfix(expression);
        System.out.println("The result of the postfix
expression is: " + result);
    } catch (Exception e) {
        System.out.println("Error in evaluation: " +
e.getMessage());
    }

    scanner.close();
  }
}
```

OUTPUT:

```
C:\Users\Atharva Mahulkar\OneDrive\Desktop\Ds_codes>java PostfixEvaluation
Enter a postfix expression (e.g., 235+4*):
34+2*
The result of the postfix expression is: 14
```

b) To implement Balancing of Parenthisis

CODE :

```
import java.util.Stack;

public class ParenthesisBalancing {

   // Function to check if
parentheses are balanced
   public static boolean
isBalanced(String expr) {
       Stack<Character> stack =
new Stack<>();

       // Traverse the string
       for (char ch :
expr.toCharArray()) {
           // Push open brackets onto
the stack
           if (ch == '(' || ch == '[' || ch
== '{') {
               stack.push(ch);
           }
           // Check for closing
brackets
           else if (ch == ')' || ch == ']' ||
ch == '}') {
               // If stack is empty, it's
unbalanced
               if (stack.isEmpty()) {
```

```
                return false;
            }
            // Pop the top element
and check if it matches
            char top = stack.pop();
            if (!isMatchingPair(top,
ch)) {
                return false;
            }
        }
    }
    // If the stack is not empty, it's
unbalanced
    return stack.isEmpty();
  }

  // Helper function to check if two
brackets are matching pairs
  private static boolean
isMatchingPair(char open, char
close) {
    return (open == '(' && close
== ')') ||
         (open == '[' && close ==
']') ||
         (open == '{' && close ==
'}');
  }

  // Main method to test the
function
  public static void main(String[]
args) {
    String expr = "{[()]}";

    if (isBalanced(expr)) {
      System.out.println("The
expression is balanced.");
    } else {
```

```
                System.out.println("The
        expression is not balanced.");
            }
        }
    }
```

OUTPUT :

```
Enter an expression with parentheses: {3+5-(25*52)[52/36]}
Parentheses are balanced.

--------------------------------
Process exited after 36.33 seconds with return value 0
Press any key to continue . . .
```

Pratical No : 5. Implement all different types of Queues

a) To Implement Circular Queue

CODE :

```
import java.util.Scanner;

public class CircularQueueWithUserInput {
  private int[] queue; // Array to store the queue elements
  private int front; // Points to the front element
  private int rear; // Points to the next insertion position
  private int size; // Size of the queue

  // Constructor to initialize the queue
  public CircularQueueWithUserInput(int size) {
    this.size = size;
    this.queue = new int[size];
    this.front = 0;
    this.rear = 0;
  }
```

```
    // Check if the queue is full
    public boolean isFull() {
        return (rear + 1) % size == front;
    }

    // Check if the queue is empty
    public boolean isEmpty() {
        return front == rear;
    }

    // Add an element to the queue
    public void enqueue(int element) {
        if (isFull()) {
            System.out.println("Queue is full. Cannot enqueue "
+ element);
            return;
        }
        queue[rear] = element;
        rear = (rear + 1) % size;
        System.out.println("Enqueued: " + element);
    }

    // Remove and return the front element from the queue
    public int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty. Cannot
dequeue.");
            return -1;
        }
        int element = queue[front];
        front = (front + 1) % size;
        System.out.println("Dequeued: " + element);
        return element;
    }

    // Peek the front element without removing it
    public int peek() {
```

```
        if (isEmpty()) {
            System.out.println("Queue is empty. Cannot peek.");
            return -1;
        }
        return queue[front];
    }

    // Display the queue elements
    public void display() {
        if (isEmpty()) {
            System.out.println("Queue is empty.");
            return;
        }
        System.out.print("Queue: ");
        int i = front;
        while (i != rear) {
            System.out.print(queue[i] + " ");
            i = (i + 1) % size;
        }
        System.out.println();
    }

    // Main method to test the Circular Queue with user input
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the queue: ");
        int size = scanner.nextInt();
        CircularQueueWithUserInput cq = new
CircularQueueWithUserInput(size + 1); // +1 to differentiate
full vs empty

        while (true) {
            System.out.println("\nChoose an operation:");
            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 the element to enqueue: ");
            int element = scanner.nextInt();
            cq.enqueue(element);
            break;
          case 2:
            cq.dequeue();
            break;
          case 3:
            int frontElement = cq.peek();
            if (frontElement != -1) {
              System.out.println("Front Element: " + frontElement);
            }
            break;
          case 4:
            cq.display();
            break;
          case 5:
            System.out.println("Exiting...");
            scanner.close();
            return;
          default:
            System.out.println("Invalid choice. Please try again.");
        }
      }
    }
}
```

OUTPUT :

```
 ---- Circular Queue Operation -----

 1. Enqueue
 2. Dequeue
 3. Display
 4. Exit
Enter the choice 1
Enter the number 21
21is inserted..
 ---- Circular Queue Operation -----

 1. Enqueue
 2. Dequeue
 3. Display
 4. Exit
Enter the choice 1
Enter the number 215
215is inserted..
 ---- Circular Queue Operation -----

 1. Enqueue
 2. Dequeue
 3. Display
 4. Exit
Enter the choice 1
Enter the number 2514
2514is inserted..
 ---- Circular Queue Operation -----
```

```
 Queue elements are
21      215     2514    321
 ---- Circular Queue Operation -----

 1. Enqueue
 2. Dequeue
 3. Display
 4. Exit
Enter the choice 2
deleted item is21
 ---- Circular Queue Operation -----

 1. Enqueue
 2. Dequeue
 3. Display
 4. Exit
Enter the choice 3

 Queue elements are
215     2514    321
 ---- Circular Queue Operation -----

 1. Enqueue
 2. Dequeue
 3. Display
 4. Exit
Enter the choice 4

--------------------------------
Process exited after 38.77 seconds with return value 0
```

Practical No : 6

Aim : Demonstrate application of queue.

a) To implement Priority Queue.

Code :

```
import java.util.PriorityQueue;
import java.util.Scanner;

class Element implements Comparable<Element> {
   int value;
   int priority;

   // Constructor
   public Element(int value, int priority) {
      this.value = value;
      this.priority = priority;
   }

   // Compare elements based on priority (higher priority comes
first)
   @Override
   public int compareTo(Element other) {
      return Integer.compare(other.priority, this.priority); // Max-
Heap style
   }

   @Override
   public String toString() {
      return "Value: " + value + ", Priority: " + priority;
   }
}

public class UserInputPriorityQueue {

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

```
    PriorityQueue<Element> priorityQueue = new
PriorityQueue<>();
    Scanner scanner = new Scanner(System.in);

    while (true) {
      System.out.println("\nChoose an operation:");
      System.out.println("1. Enqueue (Add Element)");
      System.out.println("2. Dequeue (Remove Highest
Priority)");
      System.out.println("3. Peek (View Highest Priority)");
      System.out.println("4. Display All Elements");
      System.out.println("5. Exit");
      System.out.print("Enter your choice: ");
      int choice = scanner.nextInt();

      switch (choice) {
        case 1:
          System.out.print("Enter value: ");
          int value = scanner.nextInt();
          System.out.print("Enter priority: ");
          int priority = scanner.nextInt();
          priorityQueue.add(new Element(value, priority));
          System.out.println("Enqueued: " + value + " with
priority " + priority);
          break;

        case 2:
          if (priorityQueue.isEmpty()) {
            System.out.println("Queue is empty. Cannot
dequeue.");
          } else {
            Element removed = priorityQueue.poll();
            System.out.println("Dequeued: " + removed);
          }
          break;

        case 3:
          if (priorityQueue.isEmpty()) {
            System.out.println("Queue is empty. Cannot
peek.");
```

```
                } else {
                    System.out.println("Highest Priority Element: " +
priorityQueue.peek());
                }
                break;

            case 4:
                if (priorityQueue.isEmpty()) {
                    System.out.println("Queue is empty.");
                } else {
                    System.out.println("All Elements in Priority
Queue:");
                    for (Element e : priorityQueue) {
                        System.out.println(e);
                    }
                }
                break;

            case 5:
                System.out.println("Exiting...");
                scanner.close();
                return;

            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }
  }
}
```

Output :

```
Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 1
Enter a value to enqueue: 2
Enqueued 2 into the priority queue.

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 1
Enter a value to enqueue: 5
Enqueued 5 into the priority queue.

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 1
Enter a value to enqueue: 8
Enqueued 8 into the priority queue.

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 3
Priority Queue Contents: 8 5 2

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
```

```
Enter your choice: 1
Enter a value to enqueue: 8
Enqueued 8 into the priority queue.

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 3
Priority Queue Contents: 8 5 2

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 2
Dequeued 8 from the priority queue.

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 3
Priority Queue Contents: 5 2

Priority Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Quit
Enter your choice: 4
Exiting the program.

--------------------------------
Process exited after 24.96 seconds with return value 0
Press any key to continue . . .
```

Practical No : 7

Aim : Implementation of all types of linked list.

a) To implement Single Linked List.

Code :

```
import java.util.Scanner;

class SinglyLinkedList {
    class Node {
        int data;
        Node next;

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

    private Node head = null;

    // Insert at the end
    public void insert(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
        }
        System.out.println(data + " inserted.");
    }

    // Delete a node
    public void delete(int data) {
        if (head == null) {
```

```
        System.out.println("List is empty. Nothing to delete.");
        return;
    }
    if (head.data == data) {
        head = head.next;
        System.out.println(data + " deleted.");
        return;
    }
    Node temp = head;
    while (temp.next != null && temp.next.data != data) {
        temp = temp.next;
    }
    if (temp.next == null) {
        System.out.println(data + " not found.");
    } else {
        temp.next = temp.next.next;
        System.out.println(data + " deleted.");
    }
}

// Display the list
public void display() {
    if (head == null) {
        System.out.println("List is empty.");
        return;
    }
    Node temp = head;
    System.out.print("List: ");
    while (temp != null) {
        System.out.print(temp.data + " ");
        temp = temp.next;
    }
    System.out.println();
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    SinglyLinkedList list = new SinglyLinkedList();

    while (true) {
        System.out.println("\n1. Insert");
```

```
        System.out.println("2. Delete");
        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 insert: ");
            int value = scanner.nextInt();
            list.insert(value);
            break;
          case 2:
            System.out.print("Enter value to delete: ");
            int delValue = scanner.nextInt();
            list.delete(delValue);
            break;
          case 3:
            list.display();
            break;
          case 4:
            System.out.println("Exiting...");
            scanner.close();
            return;
          default:
            System.out.println("Invalid choice. Try again.");
        }
      }}
```

OUTPUT :

```
 1.Insert
2.Display
3.Delete
4.Exit
Enter Your Choice: 1
Enter size of the LinkedList 5

Insert element-1 1

Insert element-2 2

Insert element-3 3

Insert element-4 4

Insert element-5 5

Enter Your Choice: 2

Elements of LinkedList are: 1 2 3 4 5
Enter Your Choice: 3

Enter the index of node to delete: 2

Elements of LinkedList after deletion: 1 2 4 5
Enter Your Choice: 4

--------------------------------
Process exited after 32.25 seconds with return value 0
Press any key to continue . . .
```

b) To implement Double Linked List.

Code :

```
import java.util.Scanner;

class DoublyLinkedList {
   class Node {
      int data;
      Node prev, next;

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

   private Node head = null;

   // Insert at the end
   public void insert(int data) {
      Node newNode = new Node(data);
      if (head == null) {
         head = newNode;
      } else {
         Node temp = head;
         while (temp.next != null) {
            temp = temp.next;
         }
         temp.next = newNode;
         newNode.prev = temp;
      }
      System.out.println(data + " inserted.");
   }

   // Delete a node
   public void delete(int data) {
      if (head == null) {
         System.out.println("List is empty. Nothing to
delete.");
```

```
            return;
        }
        if (head.data == data) {
            head = head.next;
            if (head != null) head.prev = null;
            System.out.println(data + " deleted.");
            return;
        }
        Node temp = head;
        while (temp != null && temp.data != data) {
            temp = temp.next;
        }
        if (temp == null) {
            System.out.println(data + " not found.");
        } else {
            if (temp.next != null) temp.next.prev = temp.prev;
            if (temp.prev != null) temp.prev.next = temp.next;
            System.out.println(data + " deleted.");
        }
    }

    // Display the list
    public void display() {
        if (head == null) {
            System.out.println("List is empty.");
            return;
        }
        Node temp = head;
        System.out.print("List: ");
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        DoublyLinkedList list = new DoublyLinkedList();
```

```
        while (true) {
            System.out.println("\n1. Insert");
            System.out.println("2. Delete");
            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 insert: ");
                    int value = scanner.nextInt();
                    list.insert(value);
                    break;
                case 2:
                    System.out.print("Enter value to delete: ");
                    int delValue = scanner.nextInt();
                    list.delete(delValue);
                    break;
                case 3:
                    list.display();
                    break;
                case 4:
                    System.out.println("Exiting...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Try
again.");
            }
        }
    }
}
```

Output :

```
2.Delete
3.DisplayList
4.Exit
Enter your choice: 1
Enter size of the LinkedList 6

Insert element-1: 5

Insert element-2: 8

Insert element-3: 3

Insert element-4: 4

Insert element-5: 9

Insert element-6: 7

Enter your choice: 3
5<-->8<-->3<-->4<-->9<-->7<-->null
Enter your choice: 2

Enter the index of node to delete: 3

Elements after deletion: 5<-->8<-->3<-->9<-->7<-->null
Enter your choice: 4

--------------------------------
Process exited after 31.62 seconds with return value 0
Press any key to continue . . .
```

c) To implement Circular Linked List.

Code :

```
import java.util.Scanner;

class CircularLinkedList {
    class Node {
        int data;
        Node next;

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

    private Node head = null;

    // Insert at the end
    public void insert(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            newNode.next = head;
        } else {
            Node temp = head;
            while (temp.next != head) {
                temp = temp.next;
            }
            temp.next = newNode;
            newNode.next = head;
        }
        System.out.println(data + " inserted.");
    }

    // Delete a node
    public void delete(int data) {
        if (head == null) {
```

```
            System.out.println("List is empty. Nothing to
delete.");
            return;
        }
        if (head.data == data && head.next == head) {
            head = null;
            System.out.println(data + " deleted.");
            return;
        }
        Node temp = head, prev = null;
        do {
            if (temp.data == data) {
                if (prev == null) {
                    Node last = head;
                    while (last.next != head) {
                        last = last.next;
                    }
                    head = head.next;
                    last.next = head;
                } else {
                    prev.next = temp.next;
                }
                System.out.println(data + " deleted.");
                return;
            }
            prev = temp;
            temp = temp.next;
        } while (temp != head);

        System.out.println(data + " not found.");
    }

    // Display the list
    public void display() {
        if (head == null) {
            System.out.println("List is empty.");
            return;
        }
        Node temp = head;
        System.out.print("List: ");
        do {
```

```
            System.out.print(temp.data + " ");
            temp = temp.next;
        } while (temp != head);
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        CircularLinkedList list = new CircularLinkedList();

        while (true) {
            System.out.println("\n1. Insert");
            System.out.println("2. Delete");
            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 insert: ");
                    int value = scanner.nextInt();
                    list.insert(value);
                    break;
                case 2:
                    System.out.print("Enter value to delete: ");
                    int delValue = scanner.nextInt();
                    list.delete(delValue);
                    break;
                case 3:
                    list.display();
                    break;
                case 4:
                    System.out.println("Exiting...");
                    scanner.close();
                    return;
                default:
                    System.out.println("Invalid choice. Try again.");
            }
        }
    }}
```

OUTPUT :

```
D:\ds_pr\CircularLL.exe

=======MENU======
1.Insert In EmptyList
2.Insert at Begin
3.Insert at End
4.Display
5.Delete
6.Exit
Enter your choice: 1

Insert element in EmptyList: 2

Enter your choice: 2

Insert element At begin: 5

Enter your choice: 2

Insert element At begin: 3

Enter your choice: 3

Insert element At End: 6

Enter your choice: 3

Insert element At End: 9

Enter your choice: 4

The circular linked list created is as follows:
9==>6==>3==>5==>2==>9


Enter your choice: 5

Enter an element to delete: 3
The node with data 3 deleted from the list

Circular linked list after deleting 3 is as follows:
5==>2==>9==>6==>5
```

Practical No : 8

Aim : Demonstrate application of linked list.

a) To implement Polynomial Addition.

Code :

```
#include <iostream> using namespace std; class Node {
public:
   int coeff, pow;
   Node *next;
 Node(int c, int p) : coeff(c),
    pow(p), next(NULL) {}
};
class Polynomial { public:
   Node *head; Polynomial()
   : head(NULL) {} void
   insertTerm(int coeff, int
   pow) {
      Node *newTerm = new Node(coeff, pow);
      if (!head) { head = newTerm; } else {
         Node *temp = head;
         while (temp->next) {
         temp = temp->next;
         }
         temp->next = newTerm;
      }
   }
```

```
Polynomial add(const Polynomial &other) const {
    Polynomial result;
    Node *temp1 = head; Node *temp2 = other.head; while (temp1 &&
    temp2) { if (temp1->pow == temp2->pow) {
    result.insertTerm(temp1->coeff + temp2->coeff, temp1->pow);
    temp1 = temp1->next; temp2 = temp2->next;
      } else if (temp1->pow > temp2->pow) {
          result.insertTerm(temp1->coeff, temp1->pow);
          temp1 = temp1->next;
      } else { result.insertTerm(temp2->coeff, temp2-
          >pow); temp2 = temp2->next;
      }
    }
    while (temp1) { result.insertTerm(temp1->coeff,
      temp1->pow); temp1 = temp1->next;
    }
    while (temp2) { result.insertTerm(temp2->coeff,
      temp2->pow); temp2 = temp2->next;
```

```
    } return result; } void display()
  const { Node *temp = head; while
  (temp) { cout << temp->coeff; if
  (temp->pow != 0) { cout << "x^" <<
  temp->pow;
      }
      temp = temp->next; if (temp) { cout << " + ";
      }
    }
    cout << endl;
  }
};

int main() { Polynomial
  poly1, poly2, result;
  poly1.insertTerm(5, 2);
  poly1.insertTerm(4, 1);
  poly1.insertTerm(2, 0);
  poly2.insertTerm(-5, 1);
  poly2.insertTerm(-5, 0);
  cout << "1st Number: ";
  poly1.display();
  cout << "2nd Number: ";
```

poly2.display(); result =

poly1.add(poly2); cout <<

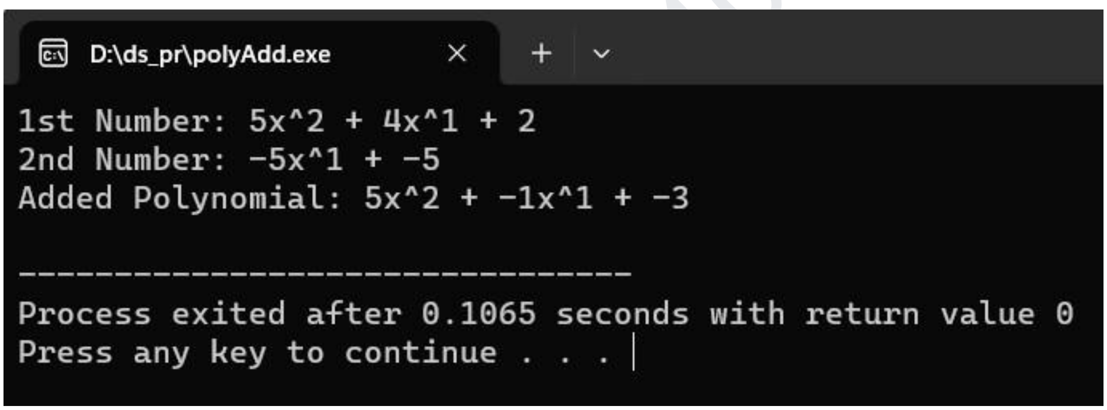
"Added Polynomial: ";

result.display();

return 0;

}

Output :



```
D:\ds_pr\polyAdd.exe
1st Number: 5x^2 + 4x^1 + 2
2nd Number: -5x^1 + -5
Added Polynomial: 5x^2 + -1x^1 + -3

--------------------------------
Process exited after 0.1065 seconds with return value 0
Press any key to continue . . .
```

b) To implement Sparse Matrix.

Code :

```
import java.util.Scanner;

class PolyNode {
    int coeff;  // Coefficient
    int exp;    // Exponent
    PolyNode next;

    PolyNode(int coeff, int exp) {
```

```
        this.coeff = coeff;
        this.exp = exp;
        this.next = null;
    }
}

public class PolynomialLinkedList {
    private PolyNode head;

    // Add a term to the polynomial
    public void addTerm(int coeff, int exp) {
        PolyNode newNode = new PolyNode(coeff, exp);
        if (head == null || head.exp < exp) {
            newNode.next = head;
            head = newNode;
        } else {
            PolyNode current = head;
            while (current.next != null && current.next.exp >= exp) {
                current = current.next;
            }
            if (current.exp == exp) {
                current.coeff += coeff; // Combine like terms
            } else {
                newNode.next = current.next;
                current.next = newNode;
            }
        }
    }

    // Display the polynomial
    public void display() {
        if (head == null) {
            System.out.println("Polynomial is empty.");
            return;
        }
        PolyNode current = head;
        while (current != null) {
            System.out.print(current.coeff + "x^" + current.exp);
            current = current.next;
            if (current != null) {
                System.out.print(" + ");
```

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

    // Add two polynomials
    public static PolynomialLinkedList addPolynomials(PolynomialLinkedList p1,
PolynomialLinkedList p2) {
        PolynomialLinkedList result = new PolynomialLinkedList();
        PolyNode n1 = p1.head;
        PolyNode n2 = p2.head;

        while (n1 != null && n2 != null) {
            if (n1.exp > n2.exp) {
                result.addTerm(n1.coeff, n1.exp);
                n1 = n1.next;
            } else if (n1.exp < n2.exp) {
                result.addTerm(n2.coeff, n2.exp);
                n2 = n2.next;
            } else {
                result.addTerm(n1.coeff + n2.coeff, n1.exp);
                n1 = n1.next;
                n2 = n2.next;
            }
        }

        while (n1 != null) {
            result.addTerm(n1.coeff, n1.exp);
            n1 = n1.next;
        }

        while (n2 != null) {
            result.addTerm(n2.coeff, n2.exp);
            n2 = n2.next;
        }

        return result;
    }

    // Evaluate the polynomial at a given value of x
    public int evaluate(int x) {
```

```
        int result = 0;
        PolyNode current = head;
        while (current != null) {
            result += current.coeff * Math.pow(x, current.exp);
            current = current.next;
        }
        return result;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        PolynomialLinkedList poly1 = new PolynomialLinkedList();
        PolynomialLinkedList poly2 = new PolynomialLinkedList();

        System.out.println("Polynomial 1:");
        while (true) {
            System.out.print("Enter coefficient (or 0 to stop): ");
            int coeff = scanner.nextInt();
            if (coeff == 0) break;
            System.out.print("Enter exponent: ");
            int exp = scanner.nextInt();
            poly1.addTerm(coeff, exp);
        }

        System.out.println("Polynomial 2:");
        while (true) {
            System.out.print("Enter coefficient (or 0 to stop): ");
            int coeff = scanner.nextInt();
            if (coeff == 0) break;
            System.out.print("Enter exponent: ");
            int exp = scanner.nextInt();
            poly2.addTerm(coeff, exp);
        }

        System.out.println("\nPolynomial 1:");
        poly1.display();
        System.out.println("Polynomial 2:");
        poly2.display();

        PolynomialLinkedList sum = addPolynomials(poly1, poly2);
        System.out.println("\nSum of the two polynomials:");
```

```
        sum.display();

        System.out.print("\nEvaluate sum polynomial at x = ");
        int x = scanner.nextInt();
        System.out.println("Result: " + sum.evaluate(x));

        scanner.close();
    }
```

Output :

```
Enter the dimensions of the matrix: 3 3
Enter the elements of the matrix: 0 0 9 5 0 8 7 0 0
The matrix is a sparse matrix
--------------------------------
Process exited after 23.39 seconds with return value 0
Press any key to continue . . .
```

Practical No : 9

Aim : Create and perform various operations on BST

a) Inserting node in BST

b) Deleting the node from BST

c) To find height of Tree

d) To perform Inorder

e) To perform Preorder

f) To perform Postorder

g) To find Maximum value of tree

Code :


```
#include<iostream> using

namespace std; struct Node{

int data;

Node *left;

Node *right;

};

Node *createNode(int value){ Node

*newNode=new Node; newNode-

>data=value; newNode->left=newNode-

>right=NULL; return newNode;

}

Node *insertNode(Node *root,int value){

if(root==NULL)

return createNode(value); if(value<root-

>data) root->left=insertNode(root-

>left,value); else if(value > root->data)

root->right=insertNode(root->right,value);

return root;

}
```

```
Node
*findMinValueNode(Node
*node){ Node
*current=node;
while(current && current-
>left!=NULL)
current=current->left;
return current;
}
Node *deleteNode(Node
*root,int value){ if(root==NULL) return
root; if(value<root->data) root-
>left=deleteNode(root->left,value); else
if(value > root->data) root-
>right=deleteNode(root->right,value);
else{ if(root->left==NULL){ Node
*temp=root->right; delete root; return
temp;
}
else if(root->right==NULL){
Node *temp=root->left;
delete root; return temp;
}
Node *temp=findMinValueNode(root->right); root-
>data=temp->data; root->right=deleteNode(root-
>right,temp->data);
```

```
} return root; } int
findHeight(Node *root){
if(root==NULL) return 0;
else{ int
leftHeight=findHeight(root-
>left); int

rightHeight=findHeight(root->right);
return max(leftHeight,rightHeight)+1;
}
} void inorderTraversal(Node
*root){ if(root!=NULL){
inorderTraversal(root->left);
cout<<root->data<<" ";
inorderTraversal(root->right);
}
} void preorderTraversal(Node
*root){ if(root!=NULL){ cout<<root-
>data<<" ";
preorderTraversal(root->left);
preorderTraversal(root->right);
}
```

```
} void
postorderTraversal(Node
*root){ if(root!=NULL){
postorderTraversal(root-
>left);
postorderTraversal(root-
>right); cout<<root-
>data<<" ";
}
} int findMaxValue(Node
*root){ if(root==NULL) return
-1; while(root->right!=NULL)
root=root->right; return root-
>data;
}
Node *buildBST(){
Node *root=NULL;
int n,value; cout<<"Enter number of
nodes in BST:"; cin>>n; for(int
i=0;i<n;++i){ cout<<"Enter value for
node"<<i+1<<":"; cin>>value;
root=insertNode(root,value);
```

```
} return root; } int main(){ Node *root=buildBST();
cout<<"\nInOrder Traversal:"; inorderTraversal(root);
cout<<endl;
cout<<"PreOrder
Traversal:";
preorderTraversal(root);
cout<<endl;
cout<<"PostOrder
Traversal:";
postorderTraversal(root);
cout<<endl; cout<<"Height
of
BST:"<<findHeight(root)<<endl; cout<<"Maximium Value in
BST:"<<findMaxValue(root)<<endl; int deleteValue;
cout<<"Enter value to delete from BST:"; cin>>deleteValue;
root=deleteNode(root,deleteValue); cout<<"BST after
deleting"<<deleteValue<<":"; inorderTraversal(root); return 0;
}
```

}

Output :

```
Enter number of nodes in BST:5
Enter value for node1:3
Enter value for node2:9
Enter value for node3:7
Enter value for node4:6
Enter value for node5:2

InOrder Traversal:2 3 6 7 9
PreOrder Traversal:3 2 9 7 6
PostOrder Traversal:2 6 7 9 3
Height of BST:4
Maximium Value in BST:9
Enter value to delete from BST:

7
BST after deleting7:2 3 6 9
--------------------------------
Process exited after 161.3 seconds with return value 0
Press any key to continue . . .
```

Practical No : 10

Aim : Implementing Heap with different operations performed

a) To perform insertion operation

b) To create Heap using Heapify method

c) To perform Heap sort

d) To delete the value in heap

Code :

```
import java.util.ArrayList;
import java.util.Scanner;

class Heap {
   private ArrayList<Integer> heap;

   public Heap() {
      heap = new ArrayList<>();
   }

   // Helper function to heapify a subtree rooted at index
   private void heapify(int index) {
      int largest = index;
      int left = 2 * index + 1;
      int right = 2 * index + 2;

      if (left < heap.size() && heap.get(left) > heap.get(largest)) {
         largest = left;
      }
      if (right < heap.size() && heap.get(right) > heap.get(largest))
{
         largest = right;
      }
      if (largest != index) {
         // Swap elements
         int temp = heap.get(index);
```

```
            heap.set(index, heap.get(largest));
            heap.set(largest, temp);

            heapify(largest);
        }
    }

    // Insert a value into the heap
    public void insert(int value) {
        heap.add(value);
        int index = heap.size() - 1;

        // Heapify-up to maintain heap property
        while (index > 0 && heap.get((index - 1) / 2) <
heap.get(index)) {
            int parent = (index - 1) / 2;
            // Swap with parent
            int temp = heap.get(index);
            heap.set(index, heap.get(parent));
            heap.set(parent, temp);

            index = parent;
        }
    }

    // Build the heap
    public void buildHeap() {
        for (int i = heap.size() / 2 - 1; i >= 0; i--) {
            heapify(i);
        }
        System.out.println("Heap Built:");
        display();
    }

    // Perform heap sort
    public void heapSort() {
        buildHeap();
        ArrayList<Integer> sorted = new ArrayList<>(heap);

        for (int i = sorted.size() - 1; i > 0; i--) {
```

```
            // Swap first and last element
            int temp = sorted.get(0);
            sorted.set(0, sorted.get(i));
            sorted.set(i, temp);

            // Remove last element and heapify
            sorted.remove(i);
            heapify(0);
        }
        System.out.println("Heap Sort Completed.");
        System.out.println("Sorted List: " + sorted);
    }

    // Delete the maximum element (root of the heap)
    public void deleteMax() {
        if (heap.isEmpty()) {
            System.out.println("Heap is Empty, cannot delete.");
            return;
        }
        // Replace root with last element
        heap.set(0, heap.get(heap.size() - 1));
        heap.remove(heap.size() - 1);

        // Heapify down to maintain heap property
        heapify(0);
    }

    // Display the heap
    public void display() {
        for (int value : heap) {
            System.out.print(value + " ");
        }
        System.out.println();
    }
}

public class HeapMain {
    public static void main(String[] args) {
        Heap maxHeap = new Heap();
        Scanner scanner = new Scanner(System.in);
```

```
int choice;

do {
    System.out.println("\n======== MENU ========");
    System.out.println("1. Insert");
    System.out.println("2. Build Heap");
    System.out.println("3. Heap Sort");
    System.out.println("4. Delete Max");
    System.out.println("5. Display");
    System.out.println("6. Exit");
    System.out.print("Enter your choice: ");
    choice = scanner.nextInt();

    switch (choice) {
        case 1:
            System.out.print("Enter value to insert: ");
            int value = scanner.nextInt();
            maxHeap.insert(value);
            break;
        case 2:
            maxHeap.buildHeap();
            break;
        case 3:
            maxHeap.heapSort();
            break;
        case 4:
            maxHeap.deleteMax();
            break;
        case 5:
            System.out.println("Heap:");
            maxHeap.display();
            break;
        case 6:
            System.out.println("Exiting Program.");
            break;
        default:
            System.out.println("Invalid Choice, please enter a
valid option.");
    }
} while (choice != 6);
```

```
        scanner.close();
    }
}
```

Output :

```
========MENU=========
1. Insert
2. Build Heap
3. Heap Sort
4. DeleteMax
5. Display
6. Exit
Enter your Choice:1
Enter value to insert:2

========MENU=========
1. Insert
2. Build Heap
3. Heap Sort
4. DeleteMax
5. Display
6. Exit
Enter your Choice:1
Enter value to insert:1

========MENU=========
1. Insert
2. Build Heap
3. Heap Sort
4. DeleteMax
5. Display
6. Exit
Enter your Choice:1
Enter value to insert:3

========MENU=========
1. Insert
```

```
========MENU=========
1. Insert
2. Build Heap
3. Heap Sort
4. DeleteMax
5. Display
6. Exit
Enter your Choice:1
Enter value to insert:4

========MENU=========
1. Insert
2. Build Heap
3. Heap Sort
4. DeleteMax
5. Display
6. Exit
Enter your Choice:1
Enter value to insert:9

========MENU=========
1. Insert
2. Build Heap
3. Heap Sort
4. DeleteMax
5. Display
6. Exit
Enter your Choice:5
Heap:9 4 2 1 3

========MENU=========
1. Insert
2. Build Heap
```

```
========MENU=========
1. Insert
2. Build Heap
3. Heap Sort
4. DeleteMax
5. Display
6. Exit
Enter your Choice:1
Enter value to insert:2

========MENU=========
1. Insert
2. Build Heap
3. Heap Sort
4. DeleteMax
5. Display
6. Exit
Enter your Choice:1
Enter value to insert:1

========MENU=========
1. Insert
2. Build Heap
3. Heap Sort
4. DeleteMax
5. Display
6. Exit
Enter your Choice:1
Enter value to insert:3

========MENU=========
1. Insert
```

```
3. Heap Sort
4. DeleteMax
5. Display
6. Exit
Enter your Choice:5
Heap:3 1 2

========MENU=========
1. Insert
2. Build Heap
3. Heap Sort
4. DeleteMax
5. Display
6. Exit
Enter your Choice:3
Heap Built:
3 1 2
Heap Sort Completed.
Sorted List:3 1 2

========MENU=========
1. Insert
2. Build Heap
3. Heap Sort
4. DeleteMax
5. Display
6. Exit
Enter your Choice:6
Exiting Program.

--------------------------------
Process exited after 166.3 seconds with return value 0
Press any key to continue . . .
```

Practical No :11

Aim : Create a graph storage structure

a) Adjacency Matrix

Code :

```
import java.util.Scanner;


class Graph {

   private int[][] adj; // Adjacency matrix

   private int nodes, edges; // Number of nodes and edges


   // Constructor to initialize the graph

   public Graph(int maxNodes) {

      adj = new int[maxNodes][maxNodes];

      nodes = 0;

      edges = 0;

   }


   // Method to create the graph

   public void createGraph() {

      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the number of nodes: ");

      nodes = scanner.nextInt();
```

```
        System.out.print("\nEnter the number of edges: ");
        edges = scanner.nextInt();

        for (int i = 1; i <= edges; i++) {
            System.out.println("\nEnter Edge " + i + ":");
            System.out.print("Enter Origin: ");
            int origin = scanner.nextInt();
            System.out.print("Enter Destination: ");
            int dest = scanner.nextInt();

            // Since it's an undirected graph, mark both [origin][dest]
and [dest][origin]
            adj[origin][dest] = 1;
            adj[dest][origin] = 1;
        }
    }

    // Method to display the adjacency matrix
    public void display() {
        System.out.println("\nAdjacency Matrix:");
        for (int i = 1; i <= nodes; i++) {
            for (int j = 1; j <= nodes; j++) {
                System.out.print(adj[i][j] + " ");
            }
```

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

public class GraphMain {
    public static void main(String[] args) {
        System.out.println("Adjacency Matrix !!!");
        final int MAX = 20;
        Graph graph = new Graph(MAX);
        graph.createGraph();
        graph.display();
    }
}
```

Output :

```
Adjacency Matrix !!!
Enter the no of nodes:4

Enter the  no of edges:4

Enter Edge:1

Enter Origin:1

Enter Dest:4


Enter Edge:2

Enter Origin:1

Enter Dest:2


Enter Edge:3

Enter Origin:1

Enter Dest:3


Enter Edge:4

Enter Origin:3

Enter Dest:2

0 1 1 1
1 0 1 0
1 1 0 0
1 0 0 0

--------------------------------
Process exited after 21.26 seconds with return value 0
Press any key to continue . . .
```

Practical No : 12

Aim : Perform various hashing techniques with Linear Probe as collision resolution scheme.

Code :

```
import java.util.Scanner;

class Graph {
  private int[][] adj; // Adjacency matrix
  private int nodes, edges; // Number of nodes and edges

  // Constructor to initialize the graph
  public Graph(int maxNodes) {
    adj = new int[maxNodes][maxNodes];
    nodes = 0;
    edges = 0;
  }

  // Method to create the graph
  public void createGraph() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the number of nodes: ");
    nodes = scanner.nextInt();

    System.out.print("\nEnter the number of edges: ");
    edges = scanner.nextInt();

    for (int i = 1; i <= edges; i++) {
      System.out.println("\nEnter Edge " + i + ":");
      System.out.print("Enter Origin: ");
      int origin = scanner.nextInt();
      System.out.print("Enter Destination: ");
      int dest = scanner.nextInt();

      // Since it's an undirected graph, mark both [origin][dest] and
[dest][origin]
      adj[origin][dest] = 1;
```

```
            adj[dest][origin] = 1;
        }
    }

    // Method to display the adjacency matrix
    public void display() {
        System.out.println("\nAdjacency Matrix:");
        for (int i = 1; i <= nodes; i++) {
            for (int j = 1; j <= nodes; j++) {
                System.out.print(adj[i][j] + " ");
            }
            System.out.println();
        }
    }
}

public class GraphMain {
    public static void main(String[] args) {
        System.out.println("Adjacency Matrix !!!");
        final int MAX = 20;
        Graph graph = new Graph(MAX);
        graph.createGraph();
        graph.display();
    }
}
```

Output :

```
Linear Probe Example !!!

1.Insert
2.Display
3.Search
4.Exit
Enter the Choice1
Enter element to be inserted:5

1.Insert
2.Display
3.Search
4.Exit
Enter the Choice1
Enter element to be inserted:2

1.Insert
2.Display
3.Search
4.Exit
Enter the Choice1
Enter element to be inserted:8

1.Insert
2.Display
3.Search
4.Exit
Enter the Choice1
Enter element to be inserted:9

1.Insert
2.Display
3.Search
4.Exit
Enter the Choice2

Elements in the hash table are
At index0       Value:0
At index1       Value:0
At index2       Value:2
```

```
3.Search
4.Exit
Enter the Choice2

Elements in the hash table are
At index0       Value:0
At index1       Value:0
At index2       Value:2
At index3       Value:0
At index4       Value:0
At index5       Value:5
At index6       Value:0
At index7       Value:0
At index8       Value:8
At index9       Value:9
1.Insert
2.Display
3.Search
4.Exit
Enter the Choice3
Enter the element to be searched:4
Value is not Found.
1.Insert
2.Display
3.Search
4.Exit
Enter the Choice3
Enter the element to be searched:5
Value is found at index:5
1.Insert
2.Display
3.Search
4.Exit
Enter the Choice4

--------------------------------
Process exited after 26.1 seconds with return value 0
Press any key to continue . . .
```

Practical No : 13

Aim : Create a minimum spanning tree using any method Kruskal's algorithm or Prim's algorithm

Code :

```
#pragma GCC optimize("O3")
#include <iostream>
#include <vector> #include
<algorithm> using
namespace std; const int
MAX = 20; int
parent[MAX]; int find(int a)
{ while (parent[a] != a) {
parent[a] =
parent[parent[a]]; a =
parent[a];
   } return a;
} void union_(int a, int b)
{ int d = find(a); int e =
find(b); parent[d] =
parent[e];
} int main() { int
vertices, edges; cout
<< "Number of
Vertices:" << endl;
cin >> vertices; cout
```

```
<< "Number of
Edges:" << endl; cin
>> edges; cout<<"\n
Enter edges in the
format 'weight src
destination' for each
edge: \n";
vector<pair<int,
pair<int, int> > >
adj; // {weight,
{source,
destination}}

  for (int i = 0; i < edges;
    i++) { int weight, src,
    destination; cin >>
    weight >> src >>
    destination;
    adj.push_back({weight,
    {src, destination}});
  } sort(adj.begin(),
  adj.end());
  for (int i = 0; i < MAX;
  i++) {
    parent[i] = i;
```

```
} vector<pair<int, int> >
tree_edges; int totalweight = 0; for
(int i = 0; i < adj.size(); i++) { int a
= adj[i].second.first; int b =
adj[i].second.second; int cost =
adj[i].first; if (find(a) != find(b)) {
totalweight += cost; union_(a, b);
tree_edges.push_back({a, b});
}
}
cout << "Edges are:" << endl; for (int i = 0; i < tree_edges.size();
i++) { cout << tree_edges[i].first << " " << tree_edges[i].second <<
endl;
}
cout<<"Total Weight of MST: ";
cout<<totalweight<<endl;
return 0;
}
```


Output :

```
Number of Vertices:
4
Number of Edges:
5

 Enter edges in the format 'weight src destination' for each edge:
1 1 2
2 2 3
3 3 4
4 1 4
5 2 4
Edges are:
1 2
2 3
3 4
Total Weight of MST: 6

--------------------------------
Process exited after 19.47 seconds with return value 0
Press any key to continue . . .
```

Practical No : 14

Aim : Implementation of Graph Traversal

a) Implement Depth First Search (DFS)

Code :

```
import java.util.*;

class KruskalMST {
    private static final int MAX = 20;
    private int[] parent = new int[MAX];

    // Find function for Union-Find (Disjoint Set Union)
    private int find(int a) {
        while (parent[a] != a) {
            parent[a] = parent[parent[a]]; // Path compression
            a = parent[a];
```

```
    }
    return a;
  }

  // Union function for Union-Find
  private void union(int a, int b) {
    int rootA = find(a);
    int rootB = find(b);
    parent[rootA] = rootB; // Attach rootA to rootB
  }

  public void kruskal() {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Number of Vertices: ");
    int vertices = scanner.nextInt();

    System.out.print("Number of Edges: ");
    int edges = scanner.nextInt();

    System.out.println("\nEnter edges in the format
'weight src destination' for each edge:");
    List<Edge> edgeList = new ArrayList<>();

    // Input edges
    for (int i = 0; i < edges; i++) {
      int weight = scanner.nextInt();
      int src = scanner.nextInt();
      int dest = scanner.nextInt();
      edgeList.add(new Edge(weight, src, dest));
    }

    // Sort edges by weight
    edgeList.sort(Comparator.comparingInt(e ->
e.weight));

    // Initialize parent array for Union-Find
    for (int i = 0; i < MAX; i++) {
      parent[i] = i;
    }
```

```
        List<Edge> mstEdges = new ArrayList<>();
        int totalWeight = 0;

        // Kruskal's Algorithm
        for (Edge edge : edgeList) {
            int a = edge.src;
            int b = edge.dest;
            int weight = edge.weight;

            if (find(a) != find(b)) {
                totalWeight += weight;
                union(a, b);
                mstEdges.add(edge);
            }
        }

        // Display MST edges
        System.out.println("Edges in the MST:");
        for (Edge edge : mstEdges) {
            System.out.println(edge.src + " - " + edge.dest);
        }

        System.out.println("Total Weight of MST: " +
totalWeight);
    }

    public static void main(String[] args) {
        KruskalMST kruskal = new KruskalMST();
        kruskal.kruskal();
    }

    // Edge class to represent edges in the graph
    static class Edge {
        int weight, src, dest;

        Edge(int weight, int src, int dest) {
            this.weight = weight;
            this.src = src;
            this.dest = dest;
        }
    }}
```

Output :

```
Enter no of Vertices:4
Enter no of Edges:5

Edges
1 4
1 2
1 3
2 3
2 4
Enter initial vertex to traverse form:2
DFS order of visited vertices:2 3 4 4
--------------------------------
Process exited after 33.61 seconds with return value 0
Press any key to continue . . .
```

b) Implement Breath First Search (BFS)

Code :

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

public class BFSGraph {
    private int[][] cost;
    private int[] visited;
    private int[] visit;
    private Queue<Integer> queue;
    private int n;

    public BFSGraph(int n) {
        this.n = n;
        cost = new int[n + 1][n + 1];
        visited = new int[n + 1];
        visit = new int[n + 1];
        queue = new LinkedList<>();
    }

    public void addEdge(int i, int j) {
        cost[i][j] = 1;
    }

    public void bfsTraversal(int startVertex) {
        System.out.println("Visited Vertices:");
        System.out.print(startVertex + " ");
        visited[startVertex] = 1;
        queue.add(startVertex);

        while (!queue.isEmpty()) {
            int currentVertex = queue.poll();

            for (int j = 1; j <= n; j++) {
                if (cost[currentVertex][j] != 0 && visited[j] != 1 && visit[j] != 1) {
                    visit[j] = 1;
                    queue.add(j);
                }
```

```
            }

            if (!queue.isEmpty()) {
                int nextVertex = queue.peek();
                System.out.print(nextVertex + " ");
                visited[nextVertex] = 1;
                visit[nextVertex] = 0;
            }
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter number of vertices: ");
        int n = scanner.nextInt();

        System.out.print("Enter number of edges: ");
        int m = scanner.nextInt();

        BFSGraph graph = new BFSGraph(n);

        System.out.println("\nEnter edges (i j):");
        for (int k = 1; k <= m; k++) {
            int i = scanner.nextInt();
            int j = scanner.nextInt();
            graph.addEdge(i, j);
        }

        System.out.print("Enter initial vertex to traverse from: ");
        int startVertex = scanner.nextInt();

        graph.bfsTraversal(startVertex);

        scanner.close();
    }
}
```

OUTPUT :

```
Enter number of vertices: 5
Enter number of edges: 6

Enter edges (i j):
1 2
1 3
2 4
3 4
3 5
4 5

Enter initial vertex to traverse from: 1
```