Practical No.1

Implementation of Sorting algorithm

Practical No.1A

Aim: - To demonstrate the bubble sort algorithm in Java

Objective: - To learn and understand the bubblesort algorithm

Source Code: -

```
import java.util.Scanner;
public class BubbleSortWithInput {
   // Method to implement bubble sort
   public static void bubbleSort(int[] array) {
      int n = array.length;
      boolean swapped;
      // Outer loop for each pass
      for (int i = 0; i < n - 1; i++) {
         swapped = false;
         // Inner loop for comparing adjacent elements
         for (int j = 0; j < n - 1 - i; j++) {
            if (array[j] > array[j + 1]) {
      // Swap the elements if they are in the wrong order
               int temp = array[j];
               array[j] = array[j + 1];
               array[j + 1] = temp;
               swapped = true;
            } }
         // If no two elements were swapped in the last
pass, the array is already sorted
         if (!swapped) {
            break;
         } }}
   // Method to print the array
   public static void printArray(int[] array) {
      for (int element : array) {
         System.out.print(element + " ");
      }
      System.out.println();
   }
   // Main method to test bubble sort with user input
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      // Input: number of elements in the array
     System.out.print("Enter the number of elements: ");
      int n = scanner.nextInt();
      // Create an array with the given size
 int[] array = new int[n];
// Input: array elements from the user
 System.out.println("Enter the elements:");
      for (int i = 0; i < n; i++) {
array[i] = scanner.nextInt();
      }
// Output: array before sorting
      System.out.println("Array before sorting:");
      printArray(array);
      // Perform bubble sort
      bubbleSort(array);
      // Output: array after sorting
      System.out.println("Array after sorting:");
      printArray(array);
      scanner.close();
   }
}
```

Output: -

```
Enter the number of elements: 5
Enter the elements:
9
76
23
12
45
Array before sorting:
9 76 23 12 45
Array after sorting:
9 12 23 45 76
```

Practical No.1B

Aim: - To demonstrate the insertion sort algorithm in Java

Objective: - To learn and understand the insertion sort algorithm

Source Code: -

```
import java.util.Scanner;
public class InsertionSort {
    // Method to perform insertion sort
    public static void insertionSort(int[] array) {
        for (int i = 1; i < array.length; i++) {
            int key = array[i];
            int j = i - 1;
// Move elements that are greater than key to one
position ahead
            while (j >= 0 && array[j] > key) {
                array[j + 1] = array[j];
                j--;
           }
            array[j + 1] = key;
        }}
    // Method to print the array
    public static void printArray(int[] array) {
        for (int value : array) {
            System.out.print(value + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Input array size
        System.out.print("Enter the number of elements
in the array: ");
        int n = scanner.nextInt();
        int[] array = new int[n];
        // Input array elements
        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            array[i] = scanner.nextInt();
        }
        // Perform insertion sort
        insertionSort(array);
        // Print sorted array
        System.out.println("Sorted array:");
        printArray(array);
// Close the scanner
        scanner.close();
    }
}
```

Output: -

```
Enter the number of elements in the array: 5
Enter the elements:
69
12
52
85
96
Sorted array:
12 52 69 85 96
```

Practical No.1C

Aim: - To demonstrate the selection sort algorithm in Java

Objective: - To learn and understand the selection sort algorithm

Source Code: -

```
import java.util.Scanner;
public class SelectionSort {
    // Method to perform selection sort
    public static void selectionSort(int[] array) {
        int n = array.length;
        for (int i = 0; i < n - 1; i++) {
 // Find the index of the minimum element in the
unsorted part
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (array[j] < array[minIndex]) {
                    minIndex = j;
                }
            }
            // Swap the found minimum element with the
first element
            int temp = array[minIndex];
            array[minIndex] = array[i];
            array[i] = temp;
        }
    }
    // Method to print the array
    public static void printArray(int[] array) {
        for (int value : array) {
            System.out.print(value + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Input array size
        System.out.print("Enter the number of elements
in the array: ");
        int n = scanner.nextInt();
        int[] array = new int[n];
        // Input array elements
        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            array[i] = scanner.nextInt();
 }
        // Perform selection sort
        selectionSort(array);
        // Print sorted array
        System.out.println("Sorted array:");
        printArray(array);
        // Close the scanner
        scanner.close();
    }
}
```

Output: -

```
Enter the number of elements in the array: 5
Enter the elements:
98
45
36
74
5
Sorted array:
5 36 45 74 98
```

Practical No.1D

Aim: - To demonstrate the shell sort algorithm in Java

Objective: - To learn and understand the shell sort algorithm

Source Code: -

```
import java.util.Scanner;
public class ShellSort {
   // Method to perform shell sort
   public static void shellSort(int[] array) {
      int n = array.length;
      // Start with a big gap, then reduce the gap
      for (int gap = n / 2; gap > 0; gap /= 2) {
         for (int i = gap; i < n; i++) {
            int temp = array[i];
            int j;
// Shift earlier gap-sorted elements up until the correct location for array[i] is found
for (j = i; j >= gap && array[j - gap] > temp; j -= gap) {
                array[j] = array[j - gap];
            }
            array[j] = temp;
         }}}
   // Method to print the array
   public static void printArray(int[] array) {
      for (int value : array) {
         System.out.print(value + " ");
      }
      System.out.println();
   }
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      // Input array size
System.out.print("Enter the number of elements in the array: ");
      int n = scanner.nextInt();
      int[] array = new int[n];
      // Input array elements
      System.out.println("Enter the elements:");
      for (int i = 0; i < n; i++) {
         array[i] = scanner.nextInt();
      }
// Perform shell sort
         shellSort(array);
      // Print sorted array
      System.out.println("Sorted array:");
      printArray(array);
      // Close the scanner
      scanner.close();
   }}
```

Output: -

```
Enter the number of elements in the array: 5
Enter the elements:
8
36
74
85
12
Sorted array:
8 12 36 74 85
```

Practical No.2

Implementation of Searching algorithm

Practical No.2A

Aim: - To demonstrate the linear search algorithm in Java

Objective: - To learn and understand the linear search algorithm

Source Code: -

```
import java.util.Scanner;
public class LinearSearch {
  public static void main(String args[])
  {
    int counter, num, item, array[];
    //To capture user input
    Scanner input = new Scanner(System.in);
    System.out.println("Enter number of elements:");
    num = input.nextInt();
    //Creating array to store the all the numbers
    array = new int[num];
    System.out.println("Enter " + num + " integers");
    //Loop to store each numbers in array
    for (counter = 0; counter < num; counter++)
      array[counter] = input.nextInt();
    System.out.println("Enter the search value:");
    item = input.nextInt();
    for (counter = 0; counter < num; counter++)
    {
      if (array[counter] == item)
      {
        System.out.println(item+" is present at location "+(counter));
        /*Item is found so to stop the search and to come out of the
         * loop use break statement.*/
        break;
      }  }
    if (counter == num)
      System.out.println(item + " doesn't exist in array.");
  }}
```

Output: -

```
Enter number of elements:
5
Enter 5 integers
78
45
32
12
96
Enter the search value:
32
32 is present at location 2
```

Practical No.2B

Aim: - To demonstrate the binary search algorithm in Java

Objective: - To learn and understand the binary search algorithm

Source Code: -

```
import java.util.Scanner;
public class BinarySearch {
    // Binary Search method
   public static int binarySearch(int[] array, int target)
{
        int left = 0;
        int right = array.length - 1;
        while (left <= right) {
 int mid = left + (right - left) / 2; // Avoid potential
overflow
            // Check if target is present at mid
            if (array[mid] == target) {
                return mid;  // Element found
            }
            // If target is greater, ignore left half
            if (array[mid] < target) {
                left = mid + 1;
            } else {
                // If target is smaller, ignore right half
                right = mid - 1;
            } }
        // Element is not present in the array
        return -1;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Taking input for the size of the array
        System.out.print("Enter the number of elements
in the array: ");
        int size = scanner.nextInt();
        // Declare and input the array
        int[] array = new int[size];
System.out.println("Enter " + size + " sorted
integers:");
        for (int i = 0; i < size; i++) {
            array[i] = scanner.nextInt();
        }

 // Taking input for the target element
        System.out.print("Enter the element to search: ");
        int target = scanner.nextInt();
        // Perform binary search
        int result = binarySearch(array, target);
        // Output the result
        if (result == -1) {
            System.out.println("Element not found");
        } else {
            System.out.println("Element found at index "
+ result);
        }
        // Closing the scanner
        scanner.close();
    }
}
```

Output: -

```
Enter the number of elements in the array: 5
Enter 5 sorted integers:
12
23
34
45
69
Enter the element to search: 45
Element found at index 3
```

Name: - **Roll No.**

Practical No.3

Implementation of Hashing Algorithm

Practical No.3A

Aim: - To implement modulo division hashing method to store the keys 55, 65,20,12,66, 26, 90 in an array of size 13. Use linear probe method to resolve any collisions.

Objective: - To learn and understand the modulo division hashing techniques.

Source Code: -

```
import java.util.Scanner;
public class ModuloDivisionHashing {
    private int[] hashTable;
    private int tableSize;
    // Constructor to initialize the hash table with a given size
    public ModuloDivisionHashing(int size) {
        tableSize = size;
        hashTable = new int[tableSize];
        // Initialize hash table with -1 to indicate empty slots
        for (int i = 0; i < tableSize; i++) {
            hashTable[i] = -1;
        }
    }
    // Hash function using modulo division
    private int hash(int key) {
        return key % tableSize;
    }
    // Insert a key into the hash table
    public void insert(int key) {
        int index = hash(key);
        // Linear probing in case of collision
        while (hashTable[index] != -1) {
            index = (index + 1) % tableSize;
        }
        hashTable[index] = key;
    }
    // Display the contents of the hash table
    public void display() {
        System.out.println("Hash Table:");
        for (int i = 0; i < tableSize; i++) {
            if (hashTable[i] != -1) {
System.out.println("Index " + i + ": " + hashTable[i]);
            } else {
System.out.println("Index " + i + ": empty");
            }
 }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Hash table size is fixed at 13
        int size = 13;
        ModuloDivisionHashing hashTable = new
ModuloDivisionHashing(size);
        // Fixed keys to be inserted
        int[] keys = {55, 65, 20, 12, 66, 26, 90};
        // Insert each key into the hash table
        System.out.println("Inserting keys: ");
        for (int key : keys) {
            System.out.println("Inserting key: " + key);
            hashTable.insert(key);
        }
        // Display the hash table
        hashTable.display();
        scanner.close();
    }
}
```

Output: -

```
Inserting keys:
Inserting key: 55
Inserting key: 65
Inserting key: 20
Inserting key: 12
Inserting key: 66
Inserting key: 26
Inserting key: 90
Hash Table:
Index 0: 65
Index 1: 66
Index 2: 26
Index 3: 55
Index 4: 90
Index 5: empty
Index 6: empty
Index 7: 20
Index 8: empty
Index 9: empty
Index 10: empty
Index 11: empty
Index 12: 12
```

Practical No.3B

Aim: - To implement digit extraction method (1, 3 and 5th) for hashing the following values 224562, 140145, 144467, 137456, 214576, 199645, 214562 in an array of 19 elements. Use linear probe method to resolve any collision.

Objective: - To learn and understand the digit extraction hashing techniques.

Source Code: -

```
import java.util.Scanner;
public class DigitExtraction {
   public int[] hashTable;
   public int tableSize;
   // Constructor to initialize the hash table with a given size
   public DigitExtraction(int size) {
      tableSize = size;
      hashTable = new int[tableSize];
      // Initialize hash table with -1 to indicate empty slots
      for (int i = 0; i < tableSize; i++) {
         hashTable[i] = -1;
      }
   }
   // Digit extraction hash function (1st, 3rd, and 5th digits)
   private int extractDigits(int key) {
      String keyStr = String.valueOf(key);
      // Extract 1st, 3rd, and 5th digits
      int extractedNumber = Integer.parseInt("" + keyStr.charAt(0) + keyStr.charAt(2) + keyStr.charAt(4));
      return extractedNumber % tableSize;
   }
   // Insert a key into the hash table
   public void insert(int key) {
      int index = extractDigits(key);
      // Linear probing in case of collision
      while (hashTable[index] != -1) {
         index = (index + 1) % tableSize;
      }
      hashTable[index] = key;
   }
// Display the contents of the hash table
   public void display() {
      System.out.println("Hash Table:");
      for (int i = 0; i < tableSize; i++) {
         if (hashTable[i] != -1) {
            System.out.println("Index " + i + ": " + hashTable[i]);
         } else {
            System.out.println("Index " + i + ": empty");
         }
      }
   }
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      // Hash table size is fixed at 19
      int size = 19;
      DigitExtraction hashTable = new DigitExtraction(size);
      // Keys to be inserted
      int[] keys = {224562, 140145, 144467, 137456, 214576, 199645, 214562};
      // Insert each key into the hash table
      System.out.println("Inserting keys: ");
      for (int key : keys) {
         System.out.println("Inserting key: " + key);
         hashTable.insert(key);
      }
      // Display the hash table
      hashTable.display();
      scanner.close();
   }
}
```

Output: -

```
Inserting keys:
Inserting key: 224562
Inserting key: 140145
Inserting key: 144467
Inserting key: 137456
Inserting key: 214576
Inserting key: 199645
Inserting key: 214562
Hash Table:
Index 0: 214576
Index 1: 214562
Index 2: empty
Index 3: empty
Index 4: 137456
Index 5: 199645
Index 6: empty
Index 7: empty
Index 8: empty
Index 9: 140145
Index 10: empty
Index 11: empty
Index 12: empty
Index 13: 144467
Index 14: empty
Index 15: empty
Index 16: empty
Index 17: empty
Index 18: 224562
```

Practical No.3C

Aim: - Write a program to implement fold boundary method for hashing the following values 224562, 140145, 144467, 137456, 214576, 199645, 214562, 162145, 234534 in an array of elements. Use linear probe method to resolve any collision.

Objective: - To learn and understand the digit extraction hashing techniques.

Source Code: -

```
import java.util.Scanner;
public class FoldBoundary {
    public int[] hashTable;
    public int tableSize;
    // Constructor to initialize the hash table with a
given size
    public FoldBoundary(int size) {
        tableSize = size;
        hashTable = new int[tableSize];
        // Initialize hash table with -1 to indicate empty
slots
for (int i = 0; i < tableSize; i++) {
            hashTable[i] = -1;
        }
}
// Fold boundary hash function
    private int foldBoundary(int key) {
        String keyStr = String.valueOf(key);

        // Split the key into two parts
        int mid = keyStr.length() / 2;
        int part1 = Integer.parseInt(keyStr.substring(0,
mid));
        int part2 =
Integer.parseInt(keyStr.substring(mid));
        // Add the two parts
        int foldedValue = part1 + part2;
        // Return modulo of folded value with table size
to get the index
        return foldedValue % tableSize;
    }
    // Insert a key into the hash table
    public void insert(int key) {
        int index = foldBoundary(key);
 // Linear probing in case of collision
        while (hashTable[index] != -1) {
            index = (index + 1) % tableSize;
        }
        hashTable[index] = key;
    }

    // Display the contents of the hash table
    public void display() {
        System.out.println("Hash Table:");
        for (int i = 0; i < tableSize; i++) {
            if (hashTable[i] != -1) {
                System.out.println("Index " + i + ": " +
hashTable[i]);
            } else {
                System.out.println("Index " + i + ": empty");
            }
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Define a hash table size
        int size = 19;
FoldBoundary hashTable = new FoldBoundary(size);
// Keys to be inserted
        int[] keys = {224562, 140145, 144467, 137456,
214576, 199645, 214562, 162145, 234534};
        // Insert each key into the hash table
        System.out.println("Inserting keys: ");
        for (int key : keys) {
            System.out.println("Inserting key: " + key);
            hashTable.insert(key);
        }
        // Display the hash table
        hashTable.display();
        scanner.close();
    }
}
```

Output: -

```
Inserting keys:
Inserting key: 224562
Inserting key: 140145
Inserting key: 144467
Inserting key: 137456
Inserting key: 214576
Inserting key: 199645
Inserting key: 214562
Inserting key: 162145
Inserting key: 234534
Hash Table:
Index 0: 140145
Index 1: empty
Index 2: empty
Index 3: 144467
Index 4: 137456
Index 5: 162145
Index 6: empty
Index 7: 224562
Index 8: 199645
Index 9: 234534
Index 10: empty
Index 11: 214576
Index 12: empty
Index 13: empty
Index 14: empty
Index 15: empty
Index 16: 214562
Index 17: empty
Index 18: empty
```

Practical No.4

Implementation of StacksPractical No.4

Aim: - To Demonstrate the working of stack, implement it as an array

Source Code:

```
import java.util.Scanner;
public class Stack {
    public int[] stackArray; // Array to store stack
elements
    public int top; // Top of the stack
    public int maxSize; // Maximum size of the stack
    // Constructor to initialize the stack
    public Stack(int size) {
        maxSize = size;
        stackArray = new int[maxSize];
        top = -1; // Stack is initially empty
    }
    // Method to add an element to the stack
    public void push(int value) {
        if (isFull()) {
            System.out.println("Stack is full. Unable to
push " + value);
        } else {
            stackArray[++top] = value;
            System.out.println(value + " pushed to
stack.");
        }
    }
    // Method to remove and return the top element of
the stack
    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack is empty. Unable to
pop.");
            return -1;
        } else {
            int poppedValue = stackArray[top--];
            System.out.println(poppedValue + " popped
from stack.");
            return poppedValue;
        }
    }


// Method to return the top element without removing
it
    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
            return -1;
        } else {
            return stackArray[top];
        }
    }
    // Method to check if the stack is empty
    public boolean isEmpty() {
        return (top == -1);
    }
    // Method to check if the stack is full
    public boolean isFull() {
        return (top == maxSize - 1);
    }
    // Method to display stack elements
    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
        } else {
            System.out.println("Stack elements:");
            for (int i = 0; i <= top; i++) {
                System.out.print(stackArray[i] + " ");
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the size of the stack: ");
        int size = scanner.nextInt();
        Stack stack = new Stack(size); // Create a stack
with user-specified size
        int choice;
        do {
            System.out.println("\nStack Operations
Menu:");
            System.out.println("1. Push");
            System.out.println("2. Pop");
            System.out.println("3. Peek");


System.out.println("4. Display");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Enter value to push: ");
```

```
                    int value = scanner.nextInt();
                    stack.push(value);
                    break;
                case 2:
                    stack.pop();
                    break;

                case 3:
                    int topElement = stack.peek();
                    if (topElement != -1) {
                        System.out.println("Top element is: " +
topElement);
                    }
                    break;
                case 4:
                    stack.display();
                    break;
                case 5:
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice. Please
try again.");
            }
        } while (choice != 5);
        scanner.close();
    }
}
```

Output:-

```
Enter the size of the stack: 5

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 10
10 pushed to stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 20
20 pushed to stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 30
30 pushed to stack.
```

```
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 4
Stack elements:
10 20 30

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 3
Top element is: 30

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 2
30 popped from stack.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 4
Stack elements:
10 20
```

Practical No.5

Name: - **Roll No.**

Implementation of Stack Applications

Practical No. 5A

Aim: - To demonstrate the conversion of infix notation to postfix notation using stack.

Source Code:-

```
import java.util.Stack;
public class InfixToPostfix {
   // Method to check if a character is an operator
   private static boolean isOperator(char ch)
 return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch
== '^');
   }
   // Method to check precedence of operators
   private static int precedence(char ch) {
      switch (ch) {
         case '+':
         case '-':
            return 1;
         case '*':
         case '/':
            return 2;
         case '^':
            return 3;
      }
      return -1;
   }
   // Method to convert infix expression to postfix
expression
   public static String infixToPostfix(String
expression) {
      StringBuilder result = new StringBuilder(); //
Resulting postfix expression
      Stack<Character> stack = new Stack<>(); //
Stack to hold operators and parentheses
      for (int i = 0; i < expression.length(); i++) {
         char ch = expression.charAt(i);
    // If the character is an operand, add it to the result
         if (Character.isLetterOrDigit(ch)) {
            result.append(ch);
         }
         // If the character is '(', push it to the stack
         else if (ch == '(') {

 stack.push(ch);}
         // If the character is ')', pop until '(' is found
else if (ch == ')') {
while (!stack.isEmpty() && stack.peek() != '(') {
               result.append(stack.pop());
            }
         if (!stack.isEmpty() && stack.peek() == '(') {
      stack.pop(); // Remove the '(' from the stack
            } }
         // If the character is an operator
         else if (isOperator(ch)) {
            while (!stack.isEmpty() && precedence(ch)
<= precedence(stack.peek())) {
               result.append(stack.pop());
            }
            stack.push(ch); // Push the current operator
to the stack
         }}
      // Pop the remaining operators from the stack
      while (!stack.isEmpty()) {
         result.append(stack.pop());
      }
      return result.toString();
   }
   // Main method to test the conversion
   public static void main(String[] args) {
      String infixExpression = "A+B*(C^D-E)";
      System.out.println("Infix Expression: " +
infixExpression);
      String postfixExpression =
infixToPostfix(infixExpression);
      System.out.println("Postfix Expression: " +
postfixExpression);
   }
}
```

Output:-

```
Infix Expression: A+B*(C^D-E)
Postfix Expression: ABCD^E-*+
```

Practical No.5B

Aim: - Write a program that use stack to evaluate postfix expression.

Source Code:-

```
import java.util.Stack;
public class PostfixEvaluation {
   // Method to evaluate the value of a postfix
expression
   public static int evaluatePostfix(String expression)
{
      // Create a stack to store operands
      Stack<Integer> stack = new Stack<>();
      // Scan the expression from left to right
      for (int i = 0; i < expression.length(); i++) {
         char ch = expression.charAt(i);
   // If the character is an operand, push it to the stack
         if (Character.isDigit(ch)) {
     stack.push(ch - '0'); // Convert char digit to integer
         }
         // If the character is an operator, pop two
operands from the stack, apply the operator, and push
the result back to the stack
         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;
                case '^':
      stack.push((int) Math.pow(operand1, operand2));
                   break;
                default:
     System.out.println("Invalid operator encountered:
" + ch);

return -1;
                }
             }
          }
      // The result will be the only value left in the
stack
      return stack.pop();
   }
   // Main method to test postfix evaluation
   public static void main(String[] args) {
      String postfixExpression = "53+82-*"; //
Example postfix expression
      System.out.println("Postfix Expression: " +
postfixExpression);
      int result = evaluatePostfix(postfixExpression);
      System.out.println("Result of evaluation: " +
result);
   }
}
```

Output:-

```
Postfix Expression: 53+82-*
Result of evaluation: 48
```

MCAL11 – Advanced Data Structures Lab

Aim: - To check if the parenthesis of an expressions are balanced using stack.

Source Code:-

```
import java.util.Stack;
public class BalancedParentheses {
  // Method to check if the given character is an opening bracket
  private static boolean isOpeningBracket(char ch) {
    return (ch == '(' || ch == '{' || ch == '[');
  }
  // Method to check if the given character is a closing bracket
  private static boolean isClosingBracket(char ch) {
    return (ch == ')' || ch == '}' || ch == ']');
  }
  // Method to check if two brackets form a matching pair
  private static boolean isMatchingPair(char open, char close) {
      return (open == '(' && close == ')') ||
           (open == '{' && close == '}') ||
           (open == '[' && close == ']');
  }
  // Method to check if the parentheses in the expression are balanced
  public static boolean areParenthesesBalanced(String expression) {
      Stack<Character> stack = new Stack<>();
      for (int i = 0; i < expression.length(); i++) {
        char ch = expression.charAt(i);
        // If it's an opening bracket, push it to the stack
        if (isOpeningBracket(ch)) {
          stack.push(ch);
        }
        // If it's a closing bracket, check if it matches the top of the stack
        else if (isClosingBracket(ch)) {
          // If the stack is empty or there's no matching opening bracket, it's unbalanced
          if (stack.isEmpty() || !isMatchingPair(stack.pop(), ch)) {
return false;  }} }
      // If the stack is empty after processing, the parentheses are balanced
      return stack.isEmpty();
  }
  // Main method to test balanced parentheses checking
  public static void main(String[] args) {
      String expression1 = "{[()]}"; // Balanced example
      String expression2 = "{[(])}"; // Unbalanced example
      System.out.println("Expression: " + expression1);
      System.out.println("Balanced: " + areParenthesesBalanced(expression1));
      System.out.println("\nExpression: " + expression2);
      System.out.println("Balanced: " + areParenthesesBalanced(expression2));
  }
}
```

Output:-

```
Expression: {[()]}
Balanced: true

Expression: {[(])}
Balanced: false
```

Practical No.6

Name: - **Roll No.**

Aim: - To demonstrate the working of an ordinary/simple queue implementing it as an array

Source Code:

```
import java.util.Scanner;
public class SimpleQueue {
   public int[] queueArray; // Array to store queue
elements
   public int maxSize; // Maximum size of the queue
   public int front; // Points to the front of the queue
   public int rear; // Points to the rear of the queue
   public int itemCount; // Number of elements in the
// Constructor to initialize the queue
   public SimpleQueue(int size) {
      maxSize = size;
      queueArray = new int[maxSize];
      front = 0; // Front is initialized to 0
      rear = -1; // Rear is initialized to -1
      itemCount = 0; // Queue is initially empty
   }
   // Method to add an element to the queue (Enqueue)
   public void enqueue(int value) {
      if (isFull()) {
         System.out.println("Queue is full. Unable to
enqueue " + value);
      } else {
         if (rear == maxSize - 1) { // Wrap around if
rear reaches the end
            rear = -1;
         }
         queueArray[++rear] = value;
         itemCount++;
         System.out.println(value + " enqueued to
queue.");
      }
   }
   // Method to remove and return the front element
from the queue (Dequeue)
   public int dequeue() {
      if (isEmpty()) {
         System.out.println("Queue is empty. Unable to
dequeue.");
         return -1;
      } else {

 int dequeuedValue = queueArray[front++];
         if (front == maxSize) { // Wrap around if front
reaches the end
            front = 0;
         }
         itemCount--;
         System.out.println(dequeuedValue + "
dequeued from queue.");
         return dequeuedValue;
      }
   }
   // Method to return the front element without
removing it (Peek)
   public int peek() {
      if (isEmpty()) {
         System.out.println("Queue is empty.");
         return -1;
      } else {
         return queueArray[front];
      }
   }
   // Method to check if the queue is empty
   public boolean isEmpty() {
      return (itemCount == 0);
   }
   // Method to check if the queue is full
   public boolean isFull() {
      return (itemCount == maxSize);
   }
   // Method to display the queue elements
   public void display() {
      if (isEmpty()) {
         System.out.println("Queue is empty.");
      } else {
         System.out.println("Queue elements:");
         int current = front;
         for (int i = 0; i < itemCount; i++) {
            System.out.print(queueArray[current] + " ");
            current = (current + 1) % maxSize; // Wrap
around using modulo
         }
         System.out.println();
      }
   }
   // Main method to demonstrate queue functionality
   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();
      SimpleQueue queue = new SimpleQueue(size); //
Create a queue with user-specified size
      int choice;
      do {
```

```
        System.out.println("\nQueue Operations
Menu:");
        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: ");
        choice = scanner.nextInt();
        switch (choice) {
          case 1:
         System.out.print("Enter value to enqueue: ");
            int value = scanner.nextInt();
            queue.enqueue(value);
            break;
          case 2:
            queue.dequeue();
            break;
          case 3:
            int frontElement = queue.peek();
            if (frontElement != -1) {
   System.out.println("Front element is: " +
frontElement);
            }
            Break;
          case 4:
            queue.display();
            break;
          case 5:
            System.out.println("Exiting...");
            break;
          default:
            System.out.println("Invalid choice. Please
try again.");
        }
      } while (choice != 5);
      scanner.close();
   }
}
```

Output:-

```
Enter the size of the queue: 5

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

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

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 30
30 enqueued to queue.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Queue elements:
10 20 30
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 3
Front element is: 10

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 2
10 dequeued from queue.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 5
Exiting...
```

Aim: - To implement circular queue using array and perform its operations

Source Code:

```
import java.util.Scanner;
public class CircularQueue {
   public int[] queueArray; // Array to store queue
elements
   public int maxSize; // Maximum size of the queue
   public int front; // Points to the front of the queue
   public int rear; // Points to the rear of the queue
   public int itemCount; // Number of elements in the
queue
   // Constructor to initialize the circular queue
   public CircularQueue(int size) {
      maxSize = size;
      queueArray = new int[maxSize];
      front = 0; // Front is initialized to 0
      rear = -1; // Rear is initialized to -1
      itemCount = 0; // Initially, the queue is empty
   }
   // Method to add an element to the circular queue
(Enqueue)
   public void enqueue(int value) {
      if (isFull()) {
         System.out.println("Queue is full. Unable to
enqueue " + value);
      } else {
         rear = (rear + 1) % maxSize; // Move rear
circularly
         queueArray[rear] = value;
         itemCount++;
         System.out.println(value + " enqueued to the
circular queue.");
      }
   }
   // Method to remove and return the front element
from the circular queue (Dequeue)
   public int dequeue() {
      if (isEmpty()) {
         System.out.println("Queue is empty. Unable to
dequeue.");
         return -1;
      } else {
         int dequeuedValue = queueArray[front];

 front = (front + 1) % maxSize; // Move front
circularly
         itemCount--;
         System.out.println(dequeuedValue + "
dequeued from the circular queue.");
         return dequeuedValue;
      }
   }
   // Method to return the front element without
removing it (Peek)
   public int peek() {
      if (isEmpty()) {
         System.out.println("Queue is empty.");
         return -1;
      } else {
         return queueArray[front];
      }
   }
   // Method to check if the circular queue is empty
   public boolean isEmpty() {
      return (itemCount == 0);
   }
   // Method to check if the circular queue is full
   public boolean isFull() {
      return (itemCount == maxSize);
   }
   // Method to display the circular queue elements
   public void display() {
      if (isEmpty()) {
         System.out.println("Queue is empty.");
      } else {
         System.out.println("Circular Queue
elements:");
         int current = front;
         for (int i = 0; i < itemCount; i++) {
            System.out.print(queueArray[current] + " ");
            current = (current + 1) % maxSize; // Move
current circularly
         }
         System.out.println();
      }
   }
   // Main method to demonstrate circular queue
functionality
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

System.out.print("Enter the size of the circular queue:
");
```

```
        int size = scanner.nextInt();
        CircularQueue queue = new CircularQueue(size);
// Create a circular queue with user-specified size
        int choice;
        do {
            System.out.println("\nCircular Queue
Operations Menu:");
            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: ");
            choice = scanner.nextInt();
            switch (choice) {
                case 1:
             System.out.print("Enter value to enqueue: ");
                    int value = scanner.nextInt();
                    queue.enqueue(value);
                    break;
                case 2:
                    queue.dequeue();
                    break;
                case 3:
                    int frontElement = queue.peek();
                    if (frontElement != -1) {
    System.out.println("Front element is: " +
frontElement);
                    }
                    break;
                case 4:
                    queue.display();
                    break;
                case 5:
                    System.out.println("Exiting...");
                    break;
                default:
System.out.println("Invalid choice. Please try again.");
            }
        } while (choice != 5);
        scanner.close();
    }
}
```

Output:-

```
Enter the size of the circular queue: 5

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 50
50 enqueued to the circular queue.

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 60
60 enqueued to the circular queue.

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 70
70 enqueued to the circular queue.

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Circular Queue elements:
50 60 70
```

```
Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 3
Front element is: 50

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 2
50 dequeued from the circular queue.

Circular Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 5
Exiting...
```

Practical No. 8

Aim: - Write a program to implement a singly linked list and perform common operations

Source Code: -

```
import java.util.Scanner;
class Node {
    int data;
    Node link;
    public Node(int data) {
        this.data = data;
        this.link = null;
    }
}
class LinkedList {
    private Node start;
    public LinkedList() {
        this.start = null;
    }
    // Create the list with multiple nodes
    public void createList() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements
to add: ");
        int n = scanner.nextInt();

        for (int i = 0; i < n; i++) {
            System.out.print("Enter the element: ");
            int data = scanner.nextInt();
            insertLast(data);
        }
    }
    // Display the list
    public void display() {
        if (start == null) {
            System.out.println("The list is empty.");
            return;
        }
        System.out.println("List elements:");
        Node temp = start;
while (temp != null) {
System.out.print(temp.data + " ");
            temp = temp.link;
}
        System.out.println(); }
    // Insert at the beginning
    public void insertBeginning(int data) {
        Node newNode = new Node(data);
        newNode.link = start;
        start = newNode;
        System.out.println("Inserted at the beginning.");
    }
    // Insert at the end
    public void insertLast(int data) {
        Node newNode = new Node(data);
        if (start == null) {
            start = newNode;
        } else {
            Node temp = start;
            while (temp.link != null) {
                temp = temp.link;
            }
            temp.link = newNode;
        }
        System.out.println("Inserted at the end.");
    }
    // Insert after a specific position
    public void insertAfter(int data, int position) {
        Node temp = start;
        for (int i = 0; i < position - 1; i++) {
            if (temp == null) {
    System.out.println("Position exceeds list length.");
                return;
            }
            temp = temp.link;
        }
        Node newNode = new Node(data);
        newNode.link = temp.link;
        temp.link = newNode;
System.out.println("Inserted at position " + (position +
1));
    }
    // Delete the first node
    public void deleteBeginning() {
        if (start == null) {
            System.out.println("The list is empty.");
            return;
        }
        start = start.link;
        System.out.println("Deleted from the
beginning.");
    }
    // Delete the last node
    public void deleteLast() {
        if (start == null) {
```

```
System.out.println("The list is empty.");
            return;
        }
        if (start.link == null) {
            start = null;
        } else {
            Node temp = start;
            while (temp.link.link != null) {
                temp = temp.link;
            }
            temp.link = null;
        }
        System.out.println("Deleted from the end.");
    }
    // Delete a specific node by value
    public void deleteElement(int data) {
        if (start == null) {
            System.out.println("The list is empty.");
            return;
        }
        if (start.data == data) {
            start = start.link;
            System.out.println("Deleted element: " + data);
            return;
        }
        Node temp = start;
        while (temp.link != null && temp.link.data !=
data) {
            temp = temp.link;
        }
if (temp.link == null) {
         System.out.println("Element not found.");
        } else {
            temp.link = temp.link.link;
            System.out.println("Deleted element: " + data);
        }
    }
public class SinglyLinkedList {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        Scanner scanner = new Scanner(System.in);
        int choice, value, position;
        do {
            System.out.println("\n***** Operations on
Singly Linked List *****");
            System.out.println("1. Create List");
            System.out.println("2. Insert Beginning");
            System.out.println("3. Insert End");
            System.out.println("4. Insert After Position");
            System.out.println("5. Delete Beginning");
            System.out.println("6. Delete Last");
            System.out.println("7. Delete Element");
            System.out.println("8. Display List");
            System.out.println("9.Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
switch (choice) {
          case 1:
          list.createList();
            break;
          case 2:
System.out.print("Enter value to insert at beginning:
");
                value = scanner.nextInt();
                list.insertBeginning(value);
                list.display();
                break;
            case 3:
    System.out.print("Enter value to insert at end: ");
                value = scanner.nextInt();
                list.insertLast(value);
                list.display();
                break;
            case 4:
               System.out.print("Enter value to insert: ");
                value = scanner.nextInt();
                System.out.print("Enter position: ");
                position = scanner.nextInt();
                list.insertAfter(value, position);
                list.display();
                break;
            case 5:
                list.deleteBeginning();
                list.display();
                break;
            case 6:
                list.deleteLast();
                list.display();
                break;
            case 7:
               System.out.print("Enter value to delete: ");
                value = scanner.nextInt();
                list.deleteElement(value);
                list.display();
                break;
            case 8:
                list.display();
                break;
```

```
case 9:
                System.out.println("Exiting.");
                break;
            default:
        System.out.println("Invalid choice. Try again.");
                break;
          }
        } while (choice != 13);
        scanner.close();
    }
}
```

Output:-

```
***** Operations on Singly Linked List *****
1. Create List
2. Insert Beginning
3. Insert End
4. Insert After Position
5. Delete Beginning
6. Delete Last
7. Delete Element
8. Display List

Enter your choice: 1
Enter the number of elements to add: 3
Enter the element: 45
Inserted at the end.
Enter the element: 23
Inserted at the end.
Enter the element: 89
Inserted at the end.

Enter your choice: 2
Enter value to insert at beginning: 12
Inserted at the beginning.
List elements:
12 45 23 89

Enter your choice: 3
Enter value to insert at end: 9
Inserted at the end.
List elements:
12 45 23 89 9

Enter your choice: 4
Enter value to insert: 98
Enter position: 3
Inserted at position 4
List elements:
12 45 23 98 89 9

Enter your choice: 5
Deleted from the beginning.
List elements:
45 23 98 89 9

Enter your choice: 6
Deleted from the end.
List elements:
45 23 98 89

Enter your choice: 7
Enter value to delete: 23
Deleted element: 23
List elements:
45 98 89

Enter your choice: 8
List elements:
45 98 89
```

Practical No. 9

Aim: - Write a program to demonstrate insert, display, search, count, reverse and sorting operation on singular linked list.

Source Code: -

```
import java.util.Scanner;
class Node {
    int data;
    Node link;
    public Node(int data) {
        this.data = data;
        this.link = null;
    }
}
class LinkedList {
    private Node start;
    public LinkedList() {
        this.start = null;
    }
    // Create the list with multiple nodes
    public void createList() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements
to add: ");
        int n = scanner.nextInt();

        for (int i = 0; i < n; i++) {
            System.out.print("Enter the element: ");
            int data = scanner.nextInt();
            insertLast(data);
        }
    }
    // Display the list
    public void display() {
        if (start == null) {
            System.out.println("The list is empty.");
            return;
        }
        System.out.println("List elements:");
        Node temp = start;
while (temp != null) {
System.out.print(temp.data + " ");
            temp = temp.link;
}

System.out.println();
    }
    // Insert at the beginning
    public void insertBeginning(int data) {
        Node newNode = new Node(data);
        newNode.link = start;
        start = newNode;
        System.out.println("Inserted at the beginning.");
    }
    // Insert at the end
    public void insertLast(int data) {
        Node newNode = new Node(data);
        if (start == null) {
            start = newNode;
        } else {
            Node temp = start;
            while (temp.link != null) {
                temp = temp.link;
            }
            temp.link = newNode;
        }
        System.out.println("Inserted at the end.");
    }
    // Insert after a specific position
    public void insertAfter(int data, int position) {
        Node temp = start;
        for (int i = 0; i < position - 1; i++) {
            if (temp == null) {
  System.out.println("Position exceeds list length.");
                return;
            }
            temp = temp.link;
        }
        Node newNode = new Node(data);
        newNode.link = temp.link;
        temp.link = newNode;
System.out.println("Inserted at position " + (position +
1));
    }
    // Delete the first node
    public void deleteBeginning() {
        if (start == null) {
            System.out.println("The list is empty.");
            return;
        }
        start = start.link;
        System.out.println("Deleted from the
beginning.");
```

```
    }
    // Delete the last node

public void deleteLast() {
        if (start == null) {
            System.out.println("The list is empty.");
            return;
        }
        if (start.link == null) {
            start = null;
        } else {
            Node temp = start;
            while (temp.link.link != null) {
                temp = temp.link;
            }
            temp.link = null;
        }
        System.out.println("Deleted from the end.");
    }
    // Delete a specific node by value
    public void deleteElement(int data) {
        if (start == null) {
            System.out.println("The list is empty.");
            return;
        }
        if (start.data == data) {
            start = start.link;
            System.out.println("Deleted element: " + data);
            return;
        }
        Node temp = start;
        while (temp.link != null && temp.link.data !=
data) {
            temp = temp.link;
        }
if (temp.link == null) {
         System.out.println("Element not found.");
        } else {
            temp.link = temp.link.link;
            System.out.println("Deleted element: " + data);
        }
    }
    // Count the number of nodes in the list
    public void countNodes() {
        int count = 0;
        Node temp = start;
        while (temp != null) {
            count++;
            temp = temp.link;
        }
System.out.println("Number of elements: " + count);}

// Reverse the list
    public void reverse() {
        Node prev = null;
        Node current = start;
        Node next;
        while (current != null) {
            next = current.link;
            current.link = prev;
            prev = current;
            current = next;
        }
        start = prev;
        System.out.println("List reversed.");
    }
    // Search for an element and return its position
    public void search(int data) {
        int position = 1;
        Node temp = start;
        while (temp != null) {
            if (temp.data == data) {
                System.out.println("Element found at
position: " + position);
                return;
            }
            temp = temp.link;
            position++;
        }
        System.out.println("Element not found.");
    }
// Sort the list
    public void sort() {
        if (start == null) {
            System.out.println("The list is empty.");
            return;
        }
        Node current = start;
        while (current != null) {
            Node index = current.link;
            while (index != null) {
                if (current.data > index.data) {
                    int temp = current.data;
                    current.data = index.data;
                    index.data = temp;
                }
                index = index.link;
```

```
        }

current = current.link;
      }
System.out.println("List sorted."); }
}
public class SinglyLinkedList {
   public static void main(String[] args) {
      LinkedList list = new LinkedList();
      Scanner scanner = new Scanner(System.in);
      int choice, value, position;
      do {
         System.out.println("\n***** Operations on
Singly Linked List *****");
         System.out.println("1. Create List");
         System.out.println("2. Insert Beginning");
         System.out.println("3. Insert End");
         System.out.println("4. Insert After Position");
         System.out.println("5. Delete Beginning");
         System.out.println("6. Delete Last");
         System.out.println("7. Delete Element");
         System.out.println("8. Display List");
         System.out.println("9. Count Elements");
         System.out.println("10. Reverse List");
         System.out.println("11. Search Element");
         System.out.println("12. Sort List");
         System.out.println("13. Exit");
         System.out.print("Enter your choice: ");
         choice = scanner.nextInt();
switch (choice) {
        case 1:
        list.createList();
         break;
        case 2:
System.out.print("Enter value to insert at beginning:
");
               value = scanner.nextInt();
               list.insertBeginning(value);
               list.display();
               break;
            case 3:
               System.out.print("Enter value to insert at
end: ");
               value = scanner.nextInt();
               list.insertLast(value);
               list.display();
               break;
            case 4:
              System.out.print("Enter value to insert: ");
               value = scanner.nextInt();

System.out.print("Enter position: ");
               position = scanner.nextInt();
               list.insertAfter(value, position);
               list.display();
               break;
            case 5:
               list.deleteBeginning();
               list.display();
               break;
            case 6:
               list.deleteLast();
               list.display();
               break;
            case 7:
              System.out.print("Enter value to delete: ");
               value = scanner.nextInt();
               list.deleteElement(value);
               list.display();
               break;
            case 8:
               list.display();
               break;
            case 9:
               list.display();
               list.countNodes();
               break;
case 10:
               list.reverse();
               list.display();
               break;
            case 11:
             System.out.print("Enter value to search: ");
               value = scanner.nextInt();
               list.search(value);
               break;
            case 12:
               list.sort();
               list.display();
               break;
            case 13:
               System.out.println("Exiting.");
               break;
            default:
         System.out.println("Invalid choice. Try again.");
               break;
         }
      } while (choice != 13);
```

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

Output:-

```
***** Operations on Singly Linked List *****
1. Create List
2. Insert Beginning
3. Insert End
4. Insert After Position
5. Delete Beginning
6. Delete Last
7. Delete Element
8. Display List
9. Count Elements
10. Reverse List
11. Search Element
12. Sort List
13. Exit
Enter your choice: 1
Enter the number of elements to add: 3
Enter the element: 45
Inserted at the end.
Enter the element: 23
Inserted at the end.
Enter the element: 89
Inserted at the end.
```

```
Enter your choice: 2
Enter value to insert at beginning: 12
Inserted at the beginning.
List elements:
12 45 23 89
```

```
Enter your choice: 3
Enter value to insert at end: 9
Inserted at the end.
List elements:
12 45 23 89 9
```

```
Enter your choice: 4
Enter value to insert: 98
Enter position: 3
Inserted at position 4
List elements:
12 45 23 98 89 9
```

```
Enter your choice: 5
Deleted from the beginning.
List elements:
45 23 98 89 9
```

```
Enter your choice: 6
Deleted from the end.
List elements:
45 23 98 89
```

```
Enter your choice: 7
Enter value to delete: 23
Deleted element: 23
List elements:
45 98 89
```

```
Enter your choice: 8
List elements:
45 98 89
```

```
Enter your choice: 9
List elements:
45 98 89
Number of elements: 3
```

```
Enter your choice: 10
List reversed.
List elements:
89 98 45
```

```
Enter your choice: 11
Enter value to search: 98
Element found at position: 2
```

```
Enter your choice: 12
List sorted.
List elements:
45 89 98
```

Practical No. 10

Aim: - To implement a circular linked list and perform its common operations.

Source Code: -

```
import java.util.Scanner;
class CNode {
    int data;
    CNode next;
    CNode prev;

    public CNode(int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}
class CircularDoublyLinkedList {
    private CNode start = null;
    private CNode last = null;
    // Create a circular doubly linked list with given
nodes
    public void create() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("How many nodes do you want
to create? ");
        int count = scanner.nextInt();
        for (int i = 0; i < count; i++)
System.out.print("Enter data for node " + (i + 1) + ":
");
            int data = scanner.nextInt();
            CNode newNode = new CNode(data);
            if (start == null) {
                start = last = newNode;
                start.next = start; // Circular link
                start.prev = start; // Doubly link to itself
            } else {
                newNode.prev = last;
                last.next = newNode;
                last = newNode;
                last.next = start;
                start.prev = last;
            }
        }
    }
// Display the elements of the circular doubly linked
list
    public void display() {
        if (start == null) {
            System.out.println("List is empty.");
            return;
        }
        CNode temp = start;
        System.out.print("List elements: ");
        do {
            System.out.print(temp.data + " ");
            temp = temp.next;
        } while (temp != start);
        System.out.println();
    }
    // Insert a node at a specified position
    public void insert() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter data to be inserted: ");
        int data = scanner.nextInt();
        CNode newNode = new CNode(data);
        System.out.println("Choose insertion option:\n1.
At the beginning\n2. At the end\n3. At a specific
position");
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                if (start == null) {
                    start = last = newNode;
                    start.next = start;
                    start.prev = start;
                } else {
                    newNode.next = start;
                    newNode.prev = last;
                    start.prev = newNode;
                    last.next = newNode;
                    start = newNode;
                }
                break;
            case 2:
                if (start == null) {
                    start = last = newNode;
                    start.next = start;
                    start.prev = start;
                } else {
                    newNode.prev = last;
                    newNode.next = start;
```

```
last.next = newNode;
                start.prev = newNode;
                    last = newNode;
                }
                break;
            case 3:
                System.out.print("Enter the position for
insertion (1-based index): ");
                int pos = scanner.nextInt();
                if (pos <= 1) {
                    System.out.println("Invalid position.");
                    break;
                }
                CNode temp = start;
                for (int i = 1; i < pos - 1 && temp.next !=
start; i++) {
                    temp = temp.next;
                }
                newNode.next = temp.next;
                newNode.prev = temp;
                temp.next.prev = newNode;
                temp.next = newNode;
                if (newNode.next == start) {
                    last = newNode;
                }
                break;
            default:
                System.out.println("Invalid choice.");
        }
    }
    // Delete a node from the list
    public void delete() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Choose deletion option:\n1.
Delete from the beginning\n2. Delete from the end\n3.
Delete at a specific position\n4. Delete all");
        int choice = scanner.nextInt();

        if (start == null) {
            System.out.println("List is empty.");
            return;
        }
        switch (choice) {
            case 1:
                System.out.println("Deleted element: " +
start.data);
if (start == last) {
                    start = last = null;
                } else {
                    start = start.next;
                    start.prev = last;
                    last.next = start;
                }
                break;
            case 2:
                System.out.println("Deleted element: " +
last.data);
                if (start == last) {
                    start = last = null;
                } else {
                    last = last.prev;
                    last.next = start;
                    start.prev = last;
                }
                break;
            case 3:
                System.out.print("Enter the position for
deletion (1-based index): ");
                int pos = scanner.nextInt();

                if (pos <= 1) {
                    System.out.println("Invalid position.");
                    break;
                }
                CNode temp = start;
                for (int i = 1; i < pos && temp.next != start;
i++) {
                    temp = temp.next;
                }
                System.out.println("Deleted element: " +
temp.data);
                temp.prev.next = temp.next;
                temp.next.prev = temp.prev;
                if (temp == start) start = temp.next;
                if (temp == last) last = temp.prev;
                break;
            case 4:
                start = last = null;
                System.out.println("All elements have been
deleted.");
                break;
            default:
                System.out.println("Invalid choice."); }}}
```

```
public class CircularLinkedList {
   public static void main(String[] args) {
      CircularDoublyLinkedList list = new CircularDoublyLinkedList();
      Scanner scanner = new Scanner(System.in);
      System.out.println("***** Operations on Circular Doubly Linked List *****");
      System.out.println("1: Create\n2: Insert\n3: Delete\n4: Display\n5: Exit");
      while (true) {
         System.out.print("Enter your choice: ");
         int choice = scanner.nextInt();
         switch (choice) {
            case 1:
               list.create();
               break;
            case 2:
               list.insert();
               break;
            case 3:
               list.delete();
               break;
            case 4:
               list.display();
               break;
            case 5:
               System.out.println("Exiting...");
               scanner.close();
               System.exit(0);
            default:
               System.out.println("Invalid choice.");
         }
      }
   }
}
```

Output: -

```
***** Operations on Circular Doubly Linked List *****
1: Create
2: Insert
3: Delete
4: Display
5: Exit
Enter your choice: 1
How many nodes do you want to create? 3
Enter data for node 1: 12
Enter data for node 2: 89
Enter data for node 3: 56
Enter your choice: 4
List elements: 12 89 56
Enter your choice: 2
Enter data to be inserted: 74
Choose insertion option:
1. At the beginning
2. At the end
3. At a specific position
1
Enter your choice: 4
List elements: 74 12 89 56
Enter your choice: 2
Enter data to be inserted: 45

Choose insertion option:
1. At the beginning
2. At the end
3. At a specific position
2
Enter your choice: 4
List elements: 74 12 89 56 45
Enter your choice: 2
Enter data to be inserted: 25
Choose insertion option:
1. At the beginning
2. At the end
3. At a specific position
3
Enter the position for insertion (1-based index): 4
Enter your choice: 4
List elements: 74 12 89 25 56 45
Enter your choice: 3
Choose deletion option:
1. Delete from the beginning
2. Delete from the end
3. Delete at a specific position
4. Delete all
1
Deleted element: 74
Enter your choice: 3
Choose deletion option:
1. Delete from the beginning
2. Delete from the end
3. Delete at a specific position
4. Delete all
2
Deleted element: 45
Enter your choice: 3
Choose deletion option:
1. Delete from the beginning
2. Delete from the end
3. Delete at a specific position
4. Delete all
3
Enter the position for deletion (1-based index): 3
Deleted element: 25
Enter your choice: 4
List elements: 12 89 56
Enter your choice: 5
Exiting...
```

Practical No. 11

Aim: - To implement a doubly linked list and perform its common operations.

Source Code: -

```
import java.util.Scanner;
class Node {
    int data;
    Node prev;
    Node next;

    public Node(int data) {
        this.data = data;
        this.prev = null;
        this.next = null;
    }
}
class DoublyLinkedList {
    private Node head;
    private Node tail;
    // Method to create nodes in the doubly linked list
    public void create(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = tail = newNode;
        } else {
            tail.next = newNode;
            newNode.prev = tail;
            tail = newNode;
        }
        System.out.println(data + " added to the list.");
    }
    // Method to display the elements of the doubly
linked list
    public void display() {
        if (head == null) {
            System.out.println("The list is empty.");
            return;
        }
        Node temp = head;
        System.out.print("List elements: ");
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }

 // Method to insert a node at a specific position in the
list
    public void insert(int data, int position) {
        Node newNode = new Node(data);
        if (position == 1) { // Insert at the beginning
            newNode.next = head;
            if (head != null) {
                head.prev = newNode;
            }
            head = newNode;
            if (tail == null) { // If list was empty
                tail = newNode;
            }
            System.out.println(data + " inserted at the
beginning.");
        } else { // Insert at a specific position or at the
end
            Node temp = head;
            int count = 1;
            while (temp != null && count < position - 1) {
                temp = temp.next;
                count++;
            }
            if (temp == null) {
                System.out.println("Position out of
bounds.");
            } else {
                newNode.next = temp.next;
                newNode.prev = temp;
                if (temp.next != null) {
                    temp.next.prev = newNode;
                } else {
                    tail = newNode; // New node is the last
node
                }
                temp.next = newNode;
                System.out.println(data + " inserted at
position " + position + ".");
            }
        }
    }
    // Method to delete a node by position
    public void delete(int position) {
        if (head == null) {
            System.out.println("The list is empty.");
```

```
            return;

}
        if (position == 1) { // Delete the first node
            System.out.println("Deleted element: " +
head.data);
            head = head.next;
            if (head != null) {
                head.prev = null;
            } else {
                tail = null; // List became empty }
 } else { // Delete a node at a specific position
            Node temp = head;
            int count = 1;
            while (temp != null && count < position) {
                temp = temp.next;
                count++;
            }
            if (temp == null) {
                System.out.println("Position out of
bounds.");
            } else {
                System.out.println("Deleted element: " +
temp.data);
                if (temp.next != null) {
                    temp.next.prev = temp.prev;
                } else {
                    tail = temp.prev; // Deleting the last node
                }
                if (temp.prev != null) {
                    temp.prev.next = temp.next;
                }}}}}
public class DoublyLinkedListDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        DoublyLinkedList list = new
DoublyLinkedList();
        int choice;
        System.out.println("***** Doubly Linked List
Operations *****");
        do {
            System.out.println("\n1: Create\n2: Insert\n3:
Display\n4: Delete\n5: Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
               System.out.print("Enter data to add to the
list: ");
                    int data = scanner.nextInt();

 list.create(data);
                    break;
                case 2:
                    System.out.print("Enter data to insert: ");
                    data = scanner.nextInt();
                    System.out.print("Enter position to insert
at (1 for beginning): ");
                    int position = scanner.nextInt();
                    list.insert(data, position);
                    break;
                case 3:
                    list.display();
                    break;
case 4:
          System.out.print("Enter position to delete from:
");
                    position = scanner.nextInt();
                    list.delete(position);
                    break;
                case 5:
                    System.out.println("Exiting...");
                    break;
                default:
         System.out.println("Invalid choice. Try again.");
                    break;
            }
        } while (choice != 5);
        scanner.close();
    }}
```

Output: -

```
***** Doubly Linked List Operations *****

1: Create
2: Insert
3: Display
4: Delete
5: Exit
Enter your choice: 1
Enter data to add to the list: 56
56 added to the list.

1: Create
2: Insert
3: Display
4: Delete
5: Exit
Enter your choice: 2
Enter data to insert: 23
Enter position to insert at (1 for beginning): 1
23 inserted at the beginning.

1: Create
2: Insert
3: Display
4: Delete
5: Exit
Enter your choice: 2
Enter data to insert: 45
Enter position to insert at (1 for beginning): 2
45 inserted at position 2.
1: Create
2: Insert
3: Display
4: Delete
5: Exit
Enter your choice: 3
List elements: 23 45 56

1: Create
2: Insert
3: Display
4: Delete
5: Exit
Enter your choice: 1
Enter data to add to the list: 89
89 added to the list.

1: Create
2: Insert
3: Display
4: Delete
5: Exit
Enter your choice: 3
List elements: 23 45 56 89

1: Create
2: Insert
3: Display
4: Delete
5: Exit
Enter your choice: 4
Enter position to delete from: 2
Deleted element: 45

1: Create
2: Insert
3: Display
4: Delete
5: Exit
Enter your choice: 2
Enter data to insert: 47
Enter position to insert at (1 for beginning): 3
47 inserted at position 3.

1: Create
2: Insert
3: Display
4: Delete
5: Exit
Enter your choice: 4
Enter position to delete from: 1
Deleted element: 23

1: Create
2: Insert
3: Display
4: Delete
5: Exit
Enter your choice: 5
Exiting...
```

Practical No. 12

Aim: - To implement a stack using linked lists and perform its common operations.

Source Code: -

```
import java.util.Scanner;
class StackusingLinkedList {
   // Node class to represent each element in the stack
   static class Node {
      int data;
      Node next;
      // Constructor to create a new node
      public Node(int data) {
         this.data = data;
         this.next = null;
      }
   }
   // Stack class that contains the linked list methods
   static class Stack {
      Node top; // Top of the stack
      public Stack() {
         top = null; // Initialize the stack as empty
      }
      // Push method to add an element to the stack
      public void push(int data) {
         Node newNode = new Node(data);
         if (top == null) {
            top = newNode; // If stack is empty, new
node becomes top
         } else {
            newNode.next = top; // New node points to
the previous top
            top = newNode; // Top points to the new
node
         }
         System.out.println(data + " pushed to the
stack.");
      }
      // Pop method to remove the top element from
the stack
      public int pop() {
         if (top == null) {
            System.out.println("Stack is empty, cannot
pop.");
            return -1; // Return -1 if the stack is empty
         }
int poppedValue = top.data;
    top = top.next; // Move the top to the next element
         return poppedValue;
      }
      // Peek method to view the top element without
removing it
      public int peek() {
         if (top == null) {
            System.out.println("Stack is empty, cannot
peek.");
            return -1; // Return -1 if the stack is empty
         }
         return top.data;
      }
      // Method to check if the stack is empty
      public boolean isEmpty() {
         return top == null;
      }
      // Method to display the stack contents
      public void display() {
         if (top == null) {
            System.out.println("Stack is empty.");
            return;
         }
         Node temp = top;
         System.out.print("Stack: ");
         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);
      Stack stack = new Stack();
      int choice;
      // Display the menu and take user input
      while (true) {
   System.out.println("\n*** Stack Operations ***");
         System.out.println("1. Push");
         System.out.println("2. Pop");
         System.out.println("3. Peek");
         System.out.println("4. Display");
         System.out.println("5. Check if Empty");
         System.out.println("6. Exit");
         System.out.print("Enter your choice: ");
```

```
 choice = scanner.nextInt();
switch (choice) {
        case 1:
          // Push operation
          System.out.print("Enter value to push: ");
          int pushValue = scanner.nextInt();
          stack.push(pushValue);
          break;
        case 2:
          // Pop operation
          int poppedValue = stack.pop();
          if (poppedValue != -1) {
System.out.println("Popped value: " + poppedValue);
          }
          break;
        case 3:
          // Peek operation
          int topValue = stack.peek();
          if (topValue != -1) {
       System.out.println("Top value: " + topValue);
          }
          break;
        case 4:
          // Display stack
          stack.display();
          break;
        case 5:
          // Check if empty
          if (stack.isEmpty()) {
            System.out.println("Stack is empty.");
          } else {
        System.out.println("Stack is not empty.");
          }
          break;
        case 6:
          // Exit
          System.out.println("Exiting...");
          scanner.close();
          return;
        default:
System.out.println("Invalid choice, please try again.");
      }
    }
  }
}
```

Output: -

```
*** Stack Operations ***
1. Push
2. Pop
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to push: 12
12 pushed to the stack.

*** Stack Operations ***
1. Push
2. Pop
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to push: 23
23 pushed to the stack.

*** Stack Operations ***
1. Push
2. Pop
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to push: 56
56 pushed to the stack.
```

```
*** Stack Operations ***
1. Push
2. Pop
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to push: 36
36 pushed to the stack.

*** Stack Operations ***
1. Push
2. Pop
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to push: 85
85 pushed to the stack.

*** Stack Operations ***
1. Push
2. Pop
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 4
Stack: 85 36 56 23 12
```

```
*** Stack Operations ***
1. Push
2. Pop
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 2
Popped value: 85

*** Stack Operations ***
1. Push
2. Pop
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 3
Top value: 36

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

*** Stack Operations ***
1. Push
2. Pop
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 4
Stack: 36 56 23 12
```

Practical No.13

Implementation of different Queue using Linked List

Practical No. 13A

Aim: - To implement an ordinary queue using a linked list and perform its common operations.

Source Code: -

```
import java.util.Scanner;
class QueueusingLinkedList {
   // Node class represents each element in the queue
   static class Node {
      int data;
      Node next;
      // Constructor to create a new node
      public Node(int data) {
         this.data = data;
         this.next = null;
      }
   }
   // Queue class containing the linked list methods
   static class Queue {
      Node front, rear; // front and rear pointers
      // Constructor to initialize the queue
      public Queue() {
         front = rear = null;
      }
   // Enqueue method to add an element to the queue
      public void enqueue(int data) {
         Node newNode = new Node(data);
         if (rear == null) {
        front = rear = newNode; // If queue is empty,
new node becomes both front and rear
  System.out.println(data + " enqueued to the queue.");
            return;
         }
         rear.next = newNode; // Add the new node at
the end of the queue
   rear = newNode; // Update the rear to the new node
 System.out.println(data + " enqueued to the queue.");
      }
      // Dequeue method to remove the front element
from the queue
      public int dequeue() {

if (front == null) {
System.out.println("Queue is empty, cannot
dequeue.");
             return -1; // Return -1 if the queue is empty
         }


 int dequeuedValue = front.data;
```

```
    front = front.next; // Move the front to the next node
        if (front == null) {
            rear = null; // If the queue becomes empty,
reset rear to null
        }
        return dequeuedValue;
    }
    // Peek method to get the front element without
removing it
    public int peek() {
        if (front == null) {
  System.out.println("Queue is empty, cannot peek.");
            return -1; // Return -1 if the queue is empty
        }
        return front.data;
    }
    // Check if the queue is empty
    public boolean isEmpty() {
        return front == null;
    }
    // Display the elements in the queue
    public void display() {
        if (front == null) {
            System.out.println("Queue is empty.");
            return;
        }
        Node temp = front;
        System.out.print("Queue: ");
        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);
    Queue queue = new Queue();
    int choice;
    // Display the menu and take user input
    while (true) {
  System.out.println("\n*** Queue Operations ***");
        System.out.println("1. Enqueue");
        System.out.println("2. Dequeue");
        System.out.println("3. Peek");
        System.out.println("4. Display");
        System.out.println("5. Check if Empty");
System.out.println("6. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextInt();
        switch (choice) {
          case 1:
            // Enqueue operation
          System.out.print("Enter value to enqueue: ");
            int enqueueValue = scanner.nextInt();
            queue.enqueue(enqueueValue);
            break;
          case 2:
            // Dequeue operation
            int dequeuedValue = queue.dequeue();
            if (dequeuedValue != -1) {
 System.out.println("Dequeued value: " +
dequeuedValue);
            }
            break;
          case 3:
            // Peek operation
            int frontValue = queue.peek();
            if (frontValue != -1) {
  System.out.println("Front value: " + frontValue);
            }
            break;
          case 4:
            // Display queue
            queue.display();
            break;
          case 5:
            // Check if empty
            if (queue.isEmpty()) {
                System.out.println("Queue is empty.");
            } else {
          System.out.println("Queue is not empty.");
            }


break;
          case 6:
            // Exit
            System.out.println("Exiting...");
            scanner.close();
            return;
          default:
 System.out.println("Invalid choice, please try
again.");
        }}}}
```

Output : -

```
*** Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to enqueue: 1
1 enqueued to the queue.

*** Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to enqueue: 2
2 enqueued to the queue.

*** Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to enqueue: 3
3 enqueued to the queue.
```

```
*** Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to enqueue: 4
4 enqueued to the queue.

*** Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to enqueue: 5
5 enqueued to the queue.

*** Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 4
Queue: 1 2 3 4 5

*** Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 2
Dequeued value: 1
```

```
*** Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 3
Front value: 2

*** Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 5
Queue is not empty.

*** Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 3
Front value: 2
```

Practical No. 13B

Aim: - To implement circular queue using a linked list and perform its common operations.

Source Code: -

```
import java.util.Scanner;

class Node {
    int data;
    Node next;

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

class CircularQueue {
    private Node tail;

    CircularQueue() {
        tail = null;
    }

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

    // Enqueue an element into the queue
    public void enqueue(int data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
            tail = newNode;
            tail.next = tail; // Points to itself
        } else {
            newNode.next = tail.next; // Link the new node
to the head
            tail.next = newNode; // Link the old tail to the
new node
            tail = newNode; // Update the tail
        }
        System.out.println("Enqueued: " + data);
    }

    // Dequeue an element from the queue
    public int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty.");
            return -1;
        }
        Node head = tail.next;
        if (head == tail) { // Single element in the queue
            tail = null;
        } else {
            tail.next = head.next; // Update the tail's next
pointer
        }
        return head.data;
    }

    // Display the queue
    public void display() {
        if (isEmpty()) {
            System.out.println("Queue is empty.");
            return;
        }
        Node current = tail.next; // Start from the head
        System.out.print("Queue: ");
        do {
            System.out.print(current.data + " ");
            current = current.next;
        } while (current != tail.next);
        System.out.println();
    }
}

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

        while (true) {
            System.out.println("\n1. 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 data to enqueue:
");
                    int data = scanner.nextInt();
                    queue.enqueue(data);
```

```
                break;
            case 2:
                int dequeued = queue.dequeue();
                if (dequeued != -1) {
                    System.out.println("Dequeued: " +
dequeued);
                }
                break;
            case 3:
                queue.display();
                break;
            case 4:
                System.out.println("Exiting...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice. Try
again.");
        }
    }
  }
}
```

Output:-

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

Enter your choice: 1
Enter data to enqueue: 20
Enqueued: 20

Enter your choice: 3
Queue: 10 20

Enter your choice: 2
Dequeued: 10

Enter your choice: 3
Queue: 20

Enter your choice: 4
Exiting...
```

Practical No. 13C

Aim: - To implement a priority queue using a linked list.

Source Code: -

```
import java.util.Scanner;
class PriorityQueueusingLinkedList {
   // Node class to represent each element in the queue
   static class Node {
      int data;  // The data to be stored (e.g., a task or a
number)
      int priority;  // The priority of the task
      Node next;

      // Constructor to create a new node
      public Node(int data, int priority) {
         this.data = data;
         this.priority = priority;
         this.next = null;
      }
   }
   // PriorityQueue class contains the linked list
methods
   static class PriorityQueue {
      Node front;  // Front of the queue
      // Constructor to initialize an empty queue
      public PriorityQueue() {
         front = null;
      }
      // Enqueue method to add an element to the
queue based on priority
      public void enqueue(int data, int priority) {
         Node newNode = new Node(data, priority);
// If the queue is empty, the new node becomes the
front
         if (front == null || front.priority < priority) {
            newNode.next = front;
            front = newNode;
            System.out.println("Enqueued: " + data + "
with priority " + priority);
            return;
         }
         // Traverse the list and find the appropriate
position for the new node
         Node temp = front;
         while (temp.next != null &&
temp.next.priority >= priority) {
            temp = temp.next;
         }
         // Insert the new node at the found position
         newNode.next = temp.next;
         temp.next = newNode;
         System.out.println("Enqueued: " + data + "
with priority " + priority);
      }
      // Dequeue method to remove the element with
the highest priority
      public int dequeue() {
         if (front == null) {
            System.out.println("Queue is empty.");
        return -1;  // Indicating that the queue is empty
         }
         // Remove the front element (highest priority)
         int dequeuedData = front.data;
         front = front.next;
         return dequeuedData;
      }
      // Peek method to get the data of the front
element without removing it
      public int peek() {
         if (front == null) {
            System.out.println("Queue is empty.");
            return -1;
         }
         return front.data;
      }
      // Display the elements of the priority queue
      public void display() {
         if (front == null) {
            System.out.println("Queue is empty.");
            return;
         }
         Node temp = front;
         System.out.print("Priority Queue: ");
         while (temp != null) {
            System.out.print("(" + temp.data + ",
Priority " + temp.priority + ") ");
            temp = temp.next;
         }
         System.out.println();
      }
     // Check if the queue is empty
      public boolean isEmpty() {
```

```
return front == null;
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        PriorityQueue queue = new PriorityQueue();
        int choice;
        // Menu for queue operations
        while (true) {
System.out.println("\n*** Priority Queue Operations
***");
            System.out.println("1. Enqueue");
            System.out.println("2. Dequeue");
            System.out.println("3. Peek");
            System.out.println("4. Display");
            System.out.println("5. Check if Empty");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    // Enqueue operation
             System.out.print("Enter value to enqueue: ");
                    int data = scanner.nextInt();
                    System.out.print("Enter priority (higher
value means higher priority): ");
                    int priority = scanner.nextInt();
                    queue.enqueue(data, priority);
                    break;
                case 2:
                    // Dequeue operation
                    int dequeuedValue = queue.dequeue();
                    if (dequeuedValue != -1) {
                        System.out.println("Dequeued value: "
+ dequeuedValue);
                    }
                    break;
                case 3:
                    // Peek operation
                    int frontValue = queue.peek();
                    if (frontValue != -1) {
                        System.out.println("Front value: " +
frontValue);
                    }
                    break;
case 4:
                    // Display the queue
                    queue.display();
                    break;
                case 5:
                    // Check if empty
                    if (queue.isEmpty()) {
                        System.out.println("Queue is empty.");
                    } else {
                 System.out.println("Queue is not empty.");
                    }
                    break;
                case 6:
                    // Exit
                    System.out.println("Exiting...");
                    scanner.close();
                    return;
                default:
System.out.println("Invalid choice, please try again.");
            }}}}
```

Output: -

```
*** Priority Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to enqueue:
4
Enter priority (higher value means higher priority): 1
Enqueued: 4 with priority 1

*** Priority Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to enqueue: 5
Enter priority (higher value means higher priority): 2
Enqueued: 5 with priority 2

*** Priority Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to enqueue: 6
Enter priority (higher value means higher priority): 3
Enqueued: 6 with priority 3
```

```
*** Priority Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to enqueue: 7
Enter priority (higher value means higher priority): 4
Enqueued: 7 with priority 4

*** Priority Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 1
Enter value to enqueue: 8
Enter priority (higher value means higher priority): 5
Enqueued: 8 with priority 5

*** Priority Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 4
Priority Queue: (8, Priority 5) (7, Priority 4) (6, Priority 3) (5, Priority 2) (4, Priority 1
```

```
*** Priority Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 2
Dequeued value: 8

*** Priority Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 3
Front value: 7

*** Priority Queue Operations ***
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Check if Empty
6. Exit
Enter your choice: 5
Queue is not empty.
```

Practical No. 13D

Aim: - To implement a double ended queue using a linked list.

Source Code: -

```
import java.util.Scanner;
class Deque {
   // Node class representing an element in the deque
   static class Node {
      int data;
      Node prev;
      Node next;
      // Constructor to create a new node
      public Node(int data) {
         this.data = data;
         this.prev = null;
         this.next = null;
      }
   }
   // Front and Rear pointers
   private Node front;
   private Node rear;
   // Constructor to initialize an empty deque
   public Deque() {
      front = null;
      rear = null;
   }
   // Insert at the front of the deque
   public void insertFront(int data) {
      Node newNode = new Node(data);
      if (front == null) { // If deque is empty
         front = rear = newNode;
      } else {
         newNode.next = front;
         front.prev = newNode;
         front = newNode;
      }
      System.out.println("Inserted " + data + " at the
front.");
   }
   // Insert at the rear of the deque
   public void insertRear(int data) {
      Node newNode = new Node(data);
      if (rear == null) { // If deque is empty
         front = rear = newNode;
      } else {
newNode.prev = rear;
         rear.next = newNode;
         rear = newNode;
      }
      System.out.println("Inserted " + data + " at the
rear.");
   }
   // Delete from the front of the deque
   public void deleteFront() {
      if (front == null) {
         System.out.println("Deque is empty.");
         return;
      }
      int deletedValue = front.data;
      if (front == rear) { // Only one element in deque
         front = rear = null;
      } else {
         front = front.next;
         front.prev = null;
      }
      System.out.println("Deleted " + deletedValue + "
from the front.");
   }
   // Delete from the rear of the deque
   public void deleteRear() {
      if (rear == null) {
         System.out.println("Deque is empty.");
         return;
      }
      int deletedValue = rear.data;
      if (front == rear) { // Only one element in deque
         front = rear = null;
      } else {
         rear = rear.prev;
         rear.next = null;
      }
      System.out.println("Deleted " + deletedValue + "
from the rear.");
   }
   // Peek at the front element of the deque
   public void peekFront() {
      if (front == null) {
         System.out.println("Deque is empty.");
         return;
      }
```

```
System.out.println("Front element is: " + front.data);
    }
    // Peek at the rear element of the deque
    public void peekRear() {
        if (rear == null) {
            System.out.println("Deque is empty.");
            return;
        }
        System.out.println("Rear element is: " +
rear.data);
    }
    // Display the elements of the deque
    public void display() {
        if (front == null) {
            System.out.println("Deque is empty.");
            return;
        }
        Node temp = front;
        System.out.print("Deque elements: ");
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
    // Check if the deque is empty
    public boolean isEmpty() {
        return front == null;
    }
}
public class DoubleEndedQueue {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Deque deque = new Deque();
        int choice;
        // Menu-driven interface for deque operations
        while (true) {
            System.out.println("\n*** Double Ended
Queue Operations ***");
            System.out.println("1. Insert at the front");
            System.out.println("2. Insert at the rear");
            System.out.println("3. Delete from the front");
            System.out.println("4. Delete from the rear");
            System.out.println("5. Peek at the front");
            System.out.println("6. Peek at the rear");
            System.out.println("7. Display the deque");
       System.out.println("8. Check if deque is empty");
            System.out.println("9. Exit");
System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    // Insert at front
    System.out.print("Enter value to insert at front: ");
                    int frontValue = scanner.nextInt();
                    deque.insertFront(frontValue);
                    break;
                case 2:
                    // Insert at rear
    System.out.print("Enter value to insert at rear: ");
                    int rearValue = scanner.nextInt();
                    deque.insertRear(rearValue);
                    break;
                case 3:
                    // Delete from front
                    deque.deleteFront();
                    break;
                case 4:
                    // Delete from rear
                    deque.deleteRear();
                    break;
                case 5:
                    // Peek at front
                    deque.peekFront();
                    break;
                case 6:
                    // Peek at rear
                    deque.peekRear();
                    break;
                case 7:
                    // Display deque
                    deque.display();
                    break;
                case 8:
                    // Check if deque is empty
                    if (deque.isEmpty()) {
                        System.out.println("Deque is empty.");
                    } else {
                System.out.println("Deque is not empty.");
                    }
                    break;
                case 9:
                    // Exit
                    System.out.println("Exiting...");
                    scanner.close();
```

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

Output: -

```
*** Double Ended Queue Operations ***
1. Insert at the front
2. Insert at the rear
3. Delete from the front
4. Delete from the rear
5. Peek at the front
6. Peek at the rear
7. Display the deque
8. Check if deque is empty
9. Exit
Enter your choice: 1
Enter value to insert at front: 9
Inserted 9 at the front.

*** Double Ended Queue Operations ***
1. Insert at the front
2. Insert at the rear
3. Delete from the front
4. Delete from the rear
5. Peek at the front
6. Peek at the rear
7. Display the deque
8. Check if deque is empty
9. Exit
Enter your choice: 2
Enter value to insert at rear: 1
Inserted 1 at the rear.

*** Double Ended Queue Operations ***
1. Insert at the front
2. Insert at the rear
3. Delete from the front
4. Delete from the rear
5. Peek at the front
6. Peek at the rear
7. Display the deque
8. Check if deque is empty
9. Exit
Enter your choice: 1
Enter value to insert at front: 5
Inserted 5 at the front.

*** Double Ended Queue Operations ***
1. Insert at the front
2. Insert at the rear
3. Delete from the front
4. Delete from the rear
5. Peek at the front
6. Peek at the rear
7. Display the deque
8. Check if deque is empty
9. Exit
Enter your choice: 2
Enter value to insert at rear: 6
Inserted 6 at the rear.

*** Double Ended Queue Operations ***
1. Insert at the front
2. Insert at the rear
3. Delete from the front
4. Delete from the rear
5. Peek at the front
6. Peek at the rear
7. Display the deque
8. Check if deque is empty
9. Exit
Enter your choice: 1
Enter value to insert at front: 3
Inserted 3 at the front.

*** Double Ended Queue Operations ***
1. Insert at the front
2. Insert at the rear
3. Delete from the front
4. Delete from the rear
5. Peek at the front
6. Peek at the rear
7. Display the deque
8. Check if deque is empty
9. Exit
Enter your choice: 7
Deque elements: 3 5 9 1 6

*** Double Ended Queue Operations ***
1. Insert at the front
2. Insert at the rear
3. Delete from the front
4. Delete from the rear
5. Peek at the front
6. Peek at the rear
7. Display the deque
8. Check if deque is empty
9. Exit
Enter your choice: 3
Deleted 3 from the front.

*** Double Ended Queue Operations ***
1. Insert at the front
2. Insert at the rear
3. Delete from the front
4. Delete from the rear
5. Peek at the front
6. Peek at the rear
7. Display the deque
8. Check if deque is empty
9. Exit
Enter your choice: 4
Deleted 6 from the rear.

*** Double Ended Queue Operations ***
1. Insert at the front
2. Insert at the rear
3. Delete from the front
4. Delete from the rear
5. Peek at the front
6. Peek at the rear
7. Display the deque
8. Check if deque is empty
9. Exit
Enter your choice: 5
Front element is: 5

*** Double Ended Queue Operations ***
1. Insert at the front
2. Insert at the rear
3. Delete from the front
4. Delete from the rear
5. Peek at the front
6. Peek at the rear
7. Display the deque
8. Check if deque is empty
9. Exit
Enter your choice: 6
Rear element is: 1

*** Double Ended Queue Operations ***
1. Insert at the front
2. Insert at the rear
3. Delete from the front
4. Delete from the rear
5. Peek at the front
6. Peek at the rear
7. Display the deque
8. Check if deque is empty
9. Exit
Enter your choice: 8
Deque is not empty.
```

Practical No. 14

Aim: - Write a program to implement Binary Search Tree, traversal (Inorder, Preorder, postorder) operations on Binary search tree.

Source Code: -

```
import java.util.Scanner;
// Node class
class Node {
   int key; // Field to store node data
   Node left, right;
   // Constructor to initialize a Node
   public Node(int key) {
      this.key = key;
      this.left = null;
      this.right = null;
   }
}
// Binary Search Tree class
class BinarySearchTree {
   Node root;
   // Constructor
   BinarySearchTree() {
      root = null;
   }
   // Insert method
   void insert(int key) {
      root = insertRec(root, key);
   }
   // Recursive method to insert a new node
   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;
   }
   // Inorder Traversal
   void inorder() {
      inorderRec(root);
   }
void inorderRec(Node root) {
      if (root != null) {
         inorderRec(root.left);
         System.out.print(root.key + " ");
         inorderRec(root.right);
      }
   }
   // Preorder Traversal
   void preorder() {
      preorderRec(root);
   }
   void preorderRec(Node root) {
      if (root != null) {
         System.out.print(root.key + " ");
         preorderRec(root.left);
         preorderRec(root.right);
      }}
   // Postorder Traversal
   void postorder() {
      postorderRec(root);
   }
   void postorderRec(Node root) {
      if (root != null) {
         postorderRec(root.left);
         postorderRec(root.right);
         System.out.print(root.key + " ");
      }}}
// Main class
public class Main {
   public static void main(String[] args) {
      BinarySearchTree bst = new BinarySearchTree();
      Scanner scanner = new Scanner(System.in);
      System.out.println("Binary Search Tree
Implementation");
      System.out.println("Enter numbers to insert into
the BST (enter -1 to stop):");
      // Insert nodes into the BST
      while (true) {
         int key = scanner.nextInt();
         if (key == -1) break;
         bst.insert(key);
      }
      while (true) {
 System.out.println("\nChoose a traversal method:");
         System.out.println("1. Inorder");
         System.out.println("2. Preorder");
         System.out.println("3. Postorder");
         System.out.println("4. Exit");
```

```
System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    System.out.println("Inorder Traversal:");
                    bst.inorder();
                    System.out.println();
                    break;
                case 2:
                    System.out.println("Preorder Traversal:");
                    bst.preorder();
                    System.out.println();
                    break;
                case 3:
                 System.out.println("Postorder Traversal:");
                    bst.postorder();
                    System.out.println();
                    break;
                case 4:
                    System.out.println("Exiting program.");
                    scanner.close();
                    return;
                default:
System.out.println("Invalid choice! Please try
again.");
            }}}}
```

Output: -

```
Binary Search Tree Implementation
Enter numbers to insert into the BST (enter -1 to stop):
50
30
70
20
40
60
80
-1

Choose a traversal method:
1. Inorder
2. Preorder
3. Postorder
4. Exit
Enter your choice: 1
Inorder Traversal:
20 30 40 50 60 70 80

Choose a traversal method:
1. Inorder
2. Preorder
3. Postorder
4. Exit
Enter your choice: 2
Preorder Traversal:
50 30 20 40 70 60 80

Choose a traversal method:
1. Inorder
2. Preorder
3. Postorder
4. Exit
Enter your choice: 3
Postorder Traversal:
20 40 30 60 80 70 50
```

Practical No. 15

Aim: - Write a program to implement various operations on Binary Search Tree.

Source Code: -

```
import java.util.Scanner;
// Node class
class Node {
   int key;
   Node left, right;
   // Constructor to initialize the key field
   public Node(int key) {
      this.key = key;
      left = right = null;
   }}
// Binary Search Tree class
class BinarySearchTree {
   Node root;
   // Constructor
   BinarySearchTree() {
      root = null;
   }
   // Insert method
   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;
   }
   // Search method
   boolean search(int key) {
      return searchRec(root, key);
   }
   boolean searchRec(Node root, int key) {
      if (root == null) {
         return false; }
if (key == root.key) {
         return true;
      }
      return key < root.key ? searchRec(root.left, key) :
searchRec(root.right, key);
   }
   // Find the smallest node
   int findMin() {
      if (root == null) {
         throw new IllegalStateException("Tree is
empty.");
      }
      return findMinRec(root).key;
   }
   Node findMinRec(Node root) {
      if (root.left == null) {
         return root;
      }
      return findMinRec(root.left);
   }
   // Find the largest node
   int findMax() {
      if (root == null) {
         throw new IllegalStateException("Tree is
empty.");
      }
      return findMaxRec(root).key;
   }
   Node findMaxRec(Node root) {
      if (root.right == null) {
         return root;
      }
      return findMaxRec(root.right);
   }
   // Count the number of nodes
   int countNodes() {
      return countNodesRec(root);
   }
   int countNodesRec(Node root) {
      if (root == null) {
         return 0;
      }
      return 1 + countNodesRec(root.left) +
countNodesRec(root.right);
   }
}
```

```
// Main class
public class MainOperation {
   public static void main(String[] args) {
       BinarySearchTree bst = new BinarySearchTree();
       Scanner scanner = new Scanner(System.in);
       System.out.println("Binary Search Tree
Operations");
       while (true) {
          System.out.println("\nChoose an operation:");
          System.out.println("1. Insert");
          System.out.println("2. Search");
          System.out.println("3. Find Smallest Node");
          System.out.println("4. Find Largest Node");
          System.out.println("5. Count Number of
Nodes");
          System.out.println("6. Exit");
          System.out.print("Enter your choice: ");
          int choice = scanner.nextInt();
          switch (choice) {
             case 1: // Insert
           System.out.print("Enter value to insert: ");
                int value = scanner.nextInt();
                bst.insert(value);
 System.out.println(value + " inserted into the BST.");
                break;
             case 2: // Search
          System.out.print("Enter value to search: ");
                int searchValue = scanner.nextInt();
                boolean found = bst.search(searchValue);
                if (found) {
System.out.println(searchValue + " exists in the
BST.");
                } else {
System.out.println(searchValue + " does not exist in
the BST.");
                }
                break;
             case 3: // Find Smallest Node
                try {
                   int smallest = bst.findMin();
    System.out.println("Smallest node: " + smallest);
                } catch (IllegalStateException e) {
                   System.out.println(e.getMessage());
                }
                break;
 case 4: // Find Largest Node
                try {
                   int largest = bst.findMax();
         System.out.println("Largest node: " + largest);
                } catch (IllegalStateException e) {
                   System.out.println(e.getMessage());
                }
                Break;
             case 5: // Count Nodes
                int count = bst.countNodes();
   System.out.println("Number of nodes: " + count);
                break;
             case 6: // Exit
                System.out.println("Exiting program.");
                scanner.close();
                return;
             default:
 System.out.println("Invalid choice! Please try
again.");
         }} }}
```

Output: -

```
Binary Search Tree Operations

Choose an operation:
1. Insert
2. Search
3. Find Smallest Node
4. Find Largest Node
5. Count Number of Nodes
6. Exit
Enter your choice: 1
Enter value to insert: 50
50 inserted into the BST.

Choose an operation:
1. Insert
2. Search
3. Find Smallest Node
4. Find Largest Node
5. Count Number of Nodes
6. Exit
Enter your choice: 1
Enter value to insert: 30
30 inserted into the BST.

Choose an operation:
1. Insert
2. Search
3. Find Smallest Node
4. Find Largest Node
5. Count Number of Nodes
6. Exit
Enter your choice: 1
Enter value to insert: 70
70 inserted into the BST.
Choose an operation:
1. Insert
2. Search
3. Find Smallest Node
4. Find Largest Node
5. Count Number of Nodes
6. Exit
Enter your choice: 1
Enter value to insert: 20
20 inserted into the BST.

Choose an operation:
1. Insert
2. Search
3. Find Smallest Node
4. Find Largest Node
5. Count Number of Nodes
6. Exit
Enter your choice: 1
Enter value to insert: 40
40 inserted into the BST.
```

```
Choose an operation:
1. Insert
2. Search
3. Find Smallest Node
4. Find Largest Node
5. Count Number of Nodes
6. Exit
Enter your choice: 2
Enter value to search: 30
30 exists in the BST.

Choose an operation:
1. Insert
2. Search
3. Find Smallest Node
4. Find Largest Node
5. Count Number of Nodes
6. Exit
Enter your choice: 3
Smallest node: 20

Choose an operation:
1. Insert
2. Search
3. Find Smallest Node
4. Find Largest Node
5. Count Number of Nodes
6. Exit
Enter your choice: 4
Largest node: 70
```

```
Choose an operation:
1. Insert
2. Search
3. Find Smallest Node
4. Find Largest Node
5. Count Number of Nodes
6. Exit
Enter your choice: 5
Number of nodes: 5

Choose an operation:
1. Insert
2. Search
3. Find Smallest Node
4. Find Largest Node
5. Count Number of Nodes
6. Exit
Enter your choice: 6
Exiting program.
```

Practical No. 16A

Aim: - Write a program to represent a MinHeap with all operations.

Source Code: -

```
import java.util.Scanner;
class Heap {
    private static final int SIZE = 15;
    private int last;
    private final int[] a;
    public Heap() {
        last = 0;
        a = new int[SIZE];
    }
    public void get() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("\nEnter size of the heap: ");
        last = scanner.nextInt();
        for (int i = 0; i < last; i++) {
            System.out.print("\nEnter the element: ");
            a[i] = scanner.nextInt();
        }
        buildHeap(a);
    }
    private void buildHeap(int[] a) {
        int walk = 0;
        while (walk < last) {
            reheapUp(a, walk);
            walk = walk + 1;
        }
    }
    private void reheapUp(int[] a, int h) {
        int parent, hold;
        if (h > 0) {
            parent = (h - 1) / 2;
            if (a[h] < a[parent]) {
                hold = a[parent];
                a[parent] = a[h];
                a[h] = hold;
                reheapUp(a, parent);
            }   } }

    public void insert(int item) {
        if (last + 1 == SIZE) {
            System.out.println("\nHeap full");
        } else {
 a[last] = item;
            reheapUp(a, last);
            last = last + 1;
            System.out.println("\nItem is inserted
successfully");
        }   }
    public void deleteNode() {
        if (last <= 0) {
            System.out.println("\nCan't delete");
        } else {
            int item = a[0];
            a[0] = a[last - 1];
            reheapDown(a, 0, last - 1);
            last = last - 1;
            System.out.println("\nThe deleted item is: " +
item);
        }   }
    private void reheapDown(int[] a, int root, int last) {
        int hold, lkey, rkey, smallerChildIndex;
        if ((root * 2 + 1) < last) {
            lkey = a[root * 2 + 1];
            if ((root * 2 + 2) < last) {
                rkey = a[root * 2 + 2];
            } else {
                rkey = Integer.MAX_VALUE;
            }
            if (lkey < rkey) {
                smallerChildIndex = root * 2 + 1;
            } else {
                smallerChildIndex = root * 2 + 2;
            }
            if (a[root] > a[smallerChildIndex]) {
                hold = a[root];
                a[root] = a[smallerChildIndex];
                a[smallerChildIndex] = hold;
                reheapDown(a, smallerChildIndex, last);
            }  }   }
    public void display() {
        System.out.println("\nCount: " + last);
        System.out.println("\nThe heap constructed is:");
        for (int i = 0; i < last; i++) {
            System.out.print("\t" + a[i]);
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Heap h = new Heap();
```

```
 h.get();
       while (true) {
          System.out.println("\n1. Insert node\n2. Delete
node\n3. Display\n4. Quit\nEnter your choice:");
          int ch = scanner.nextInt();
          switch (ch) {
             case 1:
  System.out.print("\nEnter element to be inserted: ");
                int item = scanner.nextInt();
                h.insert(item);
                break;
             case 2:
                h.deleteNode();
                break;
             case 3:
                h.display();
                break;
             case 4:
                System.out.println("Exiting...");
                System.exit(0);
                break;
             default:
                System.out.println("Invalid choice. Please
try again.");
          } } }}
```

Output:-

```
Enter size of the heap: 5

Enter the element: 4

Enter the element: 2

Enter the element: 3

Enter the element: 9

Enter the element: 8

1. Insert node
2. Delete node
3. Display
4. Quit
Enter your choice:
1

Enter element to be inserted: 6

Item is inserted successfully

1. Insert node
2. Delete node
3. Display
4. Quit
Enter your choice:
3

Count: 6

The heap constructed is:
        2       4       3       9       8       6

1. Insert node
2. Delete node
3. Display
4. Quit
Enter your choice:
2

The deleted item is: 2
```

Practical No. 16B

Aim: - Write a program to represent a MaxHeap with all operations.

Source Code: -

```
import java.util.Scanner;
class MaxHeap {
   private static final int SIZE = 15;
   private int last;
   private final int[] a;
   public MaxHeap() {
      last = 0;
      a = new int[SIZE];
      for (int i = 0; i < SIZE; i++) {
         a[i] = 0;
      }  }
   public void get() {
      Scanner scanner = new Scanner(System.in);
      System.out.print("\nEnter size of the heap: ");
      last = scanner.nextInt();
      for (int i = 0; i < last; i++) {
         System.out.print("\nEnter the element: ");
         a[i] = scanner.nextInt();
      }
      buildHeap(a);
   }
   private void buildHeap(int[] a) {
      int walk = 0;
      while (walk < last) {
         reheapUp(a, walk);
         walk = walk + 1;
      }    }
   private void reheapUp(int[] a, int h) {
      int parent, hold;
      if (h > 0) {
         parent = (h - 1) / 2;
         if (a[h] > a[parent]) {
            hold = a[parent];
            a[parent] = a[h];
            a[h] = hold;
            reheapUp(a, parent);
         }  }  }
   public void insert(int item) {
      if (last + 1 == SIZE) {
         System.out.println("\nHeap full");
      } else {
  a[last] = item;
         reheapUp(a, last);
         last = last + 1;
         System.out.println("\nItem is inserted
successfully");
      }    }
   public void deleteNode() {
      if (last <= 0) {
         System.out.println("\nCan't delete");
      } else {
         int item = a[0];
         a[0] = a[last - 1];
         last = last - 1;
         reheapDown(a, 0, last);
         System.out.println("\nThe deleted item is: " +
item);
      }  }
   private void reheapDown(int[] a, int root, int last) {
      int hold, lkey, rkey, largerChildIndex;
      if ((root * 2 + 1) < last) {
         lkey = a[root * 2 + 1];
         if ((root * 2 + 2) < last) {
            rkey = a[root * 2 + 2];
         } else {
            rkey = Integer.MIN_VALUE;
         }
         if (lkey > rkey) {
            largerChildIndex = root * 2 + 1;
         } else {
            largerChildIndex = root * 2 + 2;
         }
         if (a[root] < a[largerChildIndex]) {
            hold = a[root];
            a[root] = a[largerChildIndex];
            a[largerChildIndex] = hold;
            reheapDown(a, largerChildIndex, last);
         }    }    }
   public void display() {
      System.out.println("\nCount: " + last);
      System.out.println("\nThe heap constructed is:");
      for (int i = 0; i < last; i++) {
         System.out.print("\t" + a[i]);
      }
      System.out.println();
   }
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
```

```
MaxHeap heap = new MaxHeap();
        heap.get();
        while (true) {
            System.out.println("\n1. Insert node\n2. Delete
node\n3. Display\n4. Quit\nEnter your choice:");
            int ch = scanner.nextInt();
            switch (ch) {
                case 1:
   System.out.print("\nEnter element to be inserted: ");
                    int item = scanner.nextInt();
                    heap.insert(item);
                    break;
                case 2:
                    heap.deleteNode();
                    break;
                case 3:
                    heap.display();
                    break;
                case 4:
                    System.out.println("Exiting...");
                    System.exit(0);
                    break;
                default:
                    System.out.println("Invalid choice. Please
try again.");
            } }  }}
```

Output:-

```
Enter size of the heap: 5

Enter the element: 2

Enter the element: 9

Enter the element: 6

Enter the element: 7

Enter the element: 5

1. Insert node
2. Delete node
3. Display
4. Quit
Enter your choice:
1

Enter element to be inserted: 4

Item is inserted successfully
```

```
1. Insert node
2. Delete node
3. Display
4. Quit
Enter your choice:
3

Count: 6

The heap constructed is:
        9       7       6       2       5       4

1. Insert node
2. Delete node
3. Display
4. Quit
Enter your choice:
2

The deleted item is: 9
```

Practical No. 17

Aim: - Write a program to represent a graph using an adjacency matrix.

Source Code: -

```
import java.util.*;
public class Graph {
     private static final int MAX_VERTICES = 20; // Maximum number of vertices
    private final int[][] adjacencyMatrix; // Adjacency matrix
    private int vertexCount;
    public Graph(int vertices) {
        vertexCount = vertices;
        adjacencyMatrix = new int[MAX_VERTICES][MAX_VERTICES];
        // Initialize the adjacency matrix with 0
        for (int i = 0; i < MAX_VERTICES; i++) {
            for (int j = 0; j < MAX_VERTICES; j++) {
                adjacencyMatrix[i][j] = 0;
            }   }  }
    // Function to add an edge between vertices u and v
    public void addEdge(int u, int v) {
        if (u >= vertexCount || v >= vertexCount || u < 0 || v < 0) {
            System.out.println("Invalid edge!");
        } else {
            adjacencyMatrix[u][v] = 1;
            adjacencyMatrix[v][u] = 1; // For an undirected graph
        }    }
    // Function to display the adjacency matrix
    public void displayMatrix() {
        for (int i = 0; i < vertexCount; i++) {
            for (int j = 0; j < vertexCount; j++) {
             System.out.print(adjacencyMatrix[i][j] + " ");
            }
            System.out.println();
        }    }
    public static void main(String[] args) {
      int vertices = 6; // Number of vertices in the graph
        Graph graph = new Graph(vertices);
        // Adding edges to the graph
        graph.addEdge(0, 4);
        graph.addEdge(0, 3);
        graph.addEdge(1, 2);
        graph.addEdge(1, 4);
        graph.addEdge(1, 5);
        graph.addEdge(2, 3);
        graph.addEdge(2, 5);
        graph.addEdge(5, 3);
        graph.addEdge(5, 4);
        // Display the adjacency matrix
        graph.displayMatrix();
    }}
```

Output : -

```
0 0 0 1 1 0
0 0 1 0 1 1
0 1 0 1 0 1
1 0 1 0 0 1
1 1 0 0 0 1
0 1 1 1 1 0
```

Practical No. 18

Aim: - Write a program to implement BFS (Breadth First Search) traversal on graph.

Source Code: -

```
import java.util.*;
class BFS {
    private int vertices; // Number of vertices
    private int[][] adjMatrix; // Adjacency matrix representation of the graph
    // Constructor to initialize the graph
    public BFS(int vertices) {
        this.vertices = vertices;
        adjMatrix = new int[vertices][vertices]; // Initialize the adjacency matrix
    }
    // Method to add an edge to the graph
    public void addEdge(int src, int dest) {
        adjMatrix[src][dest] = 1; // Mark the edge in the matrix
        adjMatrix[dest][src] = 1; // Since it's an undirected graph, mark both directions
    }
    // BFS traversal starting from a given node
    public void BFS(int startVertex) {
        boolean[] visited = new boolean[vertices]; // Visited array to track visited vertices
        int[] queue = new int[vertices]; // Simulate a queue using an array
        int front = 0, rear = 0;
        // Start by marking the start vertex as visited and enqueue it
        visited[startVertex] = true;
        queue[rear++] = startVertex;
        // Process the graph
        while (front < rear) {
            int vertex = queue[front++]; // Dequeue the vertex (increment front)
            // Print the dequeued vertex
            System.out.print(vertex + " ");
   // Check all adjacent vertices of the dequeued vertex
            for (int i = 0; i < vertices; i++) {
     // If the vertex is adjacent and has not been visited
        if (adjMatrix[vertex][i] == 1 && !visited[i]) {
                visited[i] = true; // Mark as visited
queue[rear++] = i; // Enqueue the adjacent vertex
                }           }       }    }
    public static void main(String[] args) {
BFS g = new BFS(6); // Create a graph with 6 vertices
        // Adding edges to the graph
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 3);
        g.addEdge(1, 4);
        g.addEdge(2, 5);
        System.out.println("Breadth-First Search starting from vertex 0:");
        g.BFS(0); // Perform BFS starting from vertex 0
    }
}
```

Output: -

```
Breadth-First Search starting from vertex 0:
0 1 2 3 4 5
```

Practical No. 19

Aim: - Write a program to implement DFS (Depth First Search) traversal on graph.

Source Code: -

```
import java.util.*;
public class DFS {
    private int vertices; // Number of vertices
   private int[][] adjMatrix; // Adjacency matrix representation of the graph
   // Constructor to initialize the graph
   public DFS(int vertices) {
      this.vertices = vertices;
      adjMatrix = new int[vertices][vertices]; // Initialize the adjacency matrix
   }
   // Method to add an edge to the graph
   public void addEdge(int src, int dest) {
      adjMatrix[src][dest] = 1; // Mark the edge in the matrix
      adjMatrix[dest][src] = 1; // Since it's an undirected graph, mark both directions
   }
   // DFS traversal starting from a given node
   public void DFS(int startVertex) {
      boolean[] visited = new boolean[vertices]; // Visited array to track visited vertices
      // Start the DFS from the startVertex
      DFSUtil(startVertex, visited);
   }
   // Utility function for DFS (used for recursion)
   private void DFSUtil(int vertex, boolean[] visited) {
      // Mark the current vertex as visited and print it
      visited[vertex] = true;
      System.out.print(vertex + " ");
      // Recur for all adjacent vertices of the current vertex
      for (int i = 0; i < vertices; i++) {
         // If there is an edge and the vertex is not visited, call DFS recursively
         if (adjMatrix[vertex][i] == 1 && !visited[i]) {
            DFSUtil(i, visited);
         }       }  }
 public static void main(String[] args) {
  DFS g = new DFS(6); // Create a graph with 6 vertices
        // Adding edges to the graph
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 3);
        g.addEdge(1, 4);
        g.addEdge(2, 5);
System.out.println("Depth-First Search starting from vertex 0:");
        g.DFS(0); // Perform DFS starting from vertex 0
    }
}
```

Output: -

```
Depth-First Search starting from vertex 0:
0 1 3 4 2 5
```

Practical No. 20

Aim: - Write a program to find the minimum spanning tree for given graph using Kruskal's algorithm.

Source Code: -

```
import java.util.*;
// Class to represent an edge
class Edge {
   int src, dest, weight;
   // Constructor to initialize an edge
   public Edge(int src, int dest, int weight) {
      this.src = src;
      this.dest = dest;
      this.weight = weight;
   }}
// Class to represent a graph
class Graph {
   int V, E;
   Edge[] edge; // Array of edges
   // Constructor to initialize a graph with V vertices
and E edges
   public Graph(int V, int E) {
      this.V = V;
      this.E = E;
      this.edge = new Edge[E]; // Create an array of
edges
   }
}
// Class to represent a subset for union-find
class Subset {
   int parent, rank;
   // Constructor to initialize a subset
   public Subset(int parent, int rank) {
      this.parent = parent;
      this.rank = rank;
   }}
public class KruskalMST {
   // Find function with path compression
   static int find(Subset[] subsets, int i) {
      if (subsets[i].parent != i) {
         subsets[i].parent = find(subsets,
subsets[i].parent); // Path compression
      }
      return subsets[i].parent;
   }
 // Union function with union by rank
   static void union(Subset[] subsets, int x, int y) {
      int xroot = find(subsets, x);
      int yroot = find(subsets, y);
      // Union by rank
      if (subsets[xroot].rank < subsets[yroot].rank) {
         subsets[xroot].parent = yroot;
      } else if (subsets[xroot].rank >
subsets[yroot].rank) {
         subsets[yroot].parent = xroot;
      } else {
         subsets[yroot].parent = xroot;
         subsets[xroot].rank++;
      }  }
   // Comparator to sort edges based on their weight
   static class EdgeComparator implements
Comparator<Edge> {
      public int compare(Edge a, Edge b) {
         return a.weight - b.weight;
      }  }
   // Main function to construct MST using Kruskal's
algorithm
   public static void KruskalMST(Graph graph) {
      int V = graph.V;
      Edge[] result = new Edge[V]; // Array to store
the resultant MST
      int e = 0; // Index for result[]
      int i = 0; // Index for sorted edges
      // Step 1: Sort all the edges in non-decreasing
order of their weight
      Arrays.sort(graph.edge, new EdgeComparator());
      // Create V subsets for union-find
      Subset[] subsets = new Subset[V];
      for (int v = 0; v < V; ++v) {
         subsets[v] = new Subset(v, 0); // Initialize each
subset with parent as itself
      }
      // Step 2: Pick the smallest edge and increment
the index for next iteration
      while (e < V - 1 && i < graph.E) {
         Edge nextEdge = graph.edge[i++]; // Pick the
smallest edge
         int x = find(subsets, nextEdge.src);
         int y = find(subsets, nextEdge.dest);
 // If including this edge does not form a cycle, include
it in the result
         if (x != y) {
            result[e++] = nextEdge;
```

```
union(subsets, x, y);
          }      }
        // Print the resulting MST
        System.out.println("Following are the edges in
the constructed MST:");
        int minimumCost = 0;
        for (i = 0; i < e; ++i) {
            System.out.println(result[i].src + " -- " +
result[i].dest + " == " + result[i].weight);
            minimumCost += result[i].weight;
        }
        System.out.println("Minimum Cost Spanning
Tree: " + minimumCost);
    }
    public static void main(String[] args) {
        int V = 6; // Number of vertices in graph
        int E = 8; // Number of edges in graph
        Graph g = new Graph(V, E);
        // Add edges
        g.edge[0] = new Edge(0, 1, 4);
        g.edge[1] = new Edge(0, 5, 2);
        g.edge[2] = new Edge(1, 2, 6);
        g.edge[3] = new Edge(2, 3, 3);
        g.edge[4] = new Edge(3, 4, 2);
        g.edge[5] = new Edge(4, 5, 4);
        g.edge[6] = new Edge(5, 1, 5);
        g.edge[7] = new Edge(5, 2, 1);
        // Function call to find MST using Kruskal's
algorithm
        KruskalMST(g);
    }}
```

Output: -

```
Following are the edges in the constructed MST:
5 -- 2 == 1
0 -- 5 == 2
3 -- 4 == 2
2 -- 3 == 3
0 -- 1 == 4
Minimum Cost Spanning Tree: 12
```

Practical No. 20

Aim: - Write a program to find the minimum spanning tree for given graph using Prim's algorithm.

Source Code: -

```
import java.util.*;
public class PrimMST {
static final int V = 5; // Number of vertices in the graph
// Function to find the vertex with the minimum key value
   static int minKey(int key[], Boolean mstSet[]) {
      int min = Integer.MAX_VALUE;
      int minIndex = -1;
      for (int v = 0; v < V; v++) {
         if (!mstSet[v] && key[v] < min) {
            min = key[v];
            minIndex = v;
         }      }
      return minIndex;
   }
   // Function to implement Prim's MST algorithm
   static void primMST(int graph[][]) {
      int parent[] = new int[V]; // Array to store constructed MST
      int key[] = new int[V]; // Key values used to pick minimum weight edge
      Boolean mstSet[] = new Boolean[V]; // To represent vertices not yet included in MST
      // Initialize all keys to infinity and mstSet[] to false
      Arrays.fill(key, Integer.MAX_VALUE);
      Arrays.fill(mstSet, false);
      // Start with the first vertex
      key[0] = 0;
      parent[0] = -1; // First node is always the root of MST
      // Find the MST of the graph
      for (int count = 0; count < V - 1; count++) {
         // Pick the minimum key vertex from the set of vertices not yet processed
         int u = minKey(key, mstSet);
         mstSet[u] = true; // Add the picked vertex to the MST
         // Update the key value and parent index of the adjacent vertices
for (int v = 0; v < V; v++) {
            if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] < key[v]) {
               parent[v] = u;
               key[v] = graph[u][v];
            }       }      }
      // Print the constructed MST
      printMST(parent, graph);
   }
   // Function to print the MST stored in parent[]
   static void printMST(int parent[], int graph[][]) {
      System.out.println("Edge \tWeight");
      for (int i = 1; i < V; i++) {
         System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
      }
   }
   public static void main(String[] args) {
      // Graph represented as an adjacency matrix
      int graph[][] = new int[][] {
         {0, 2, 0, 6, 0},
         {2, 0, 3, 8, 5},
         {0, 3, 0, 0, 7},
         {6, 8, 0, 0, 9},
         {0, 5, 7, 9, 0}
      };
    // Call Prim's algorithm to find the MST
      primMST(graph);
   }
}
```

Output: -

```
Edge     Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```