1.Sorting and Searching Techniques

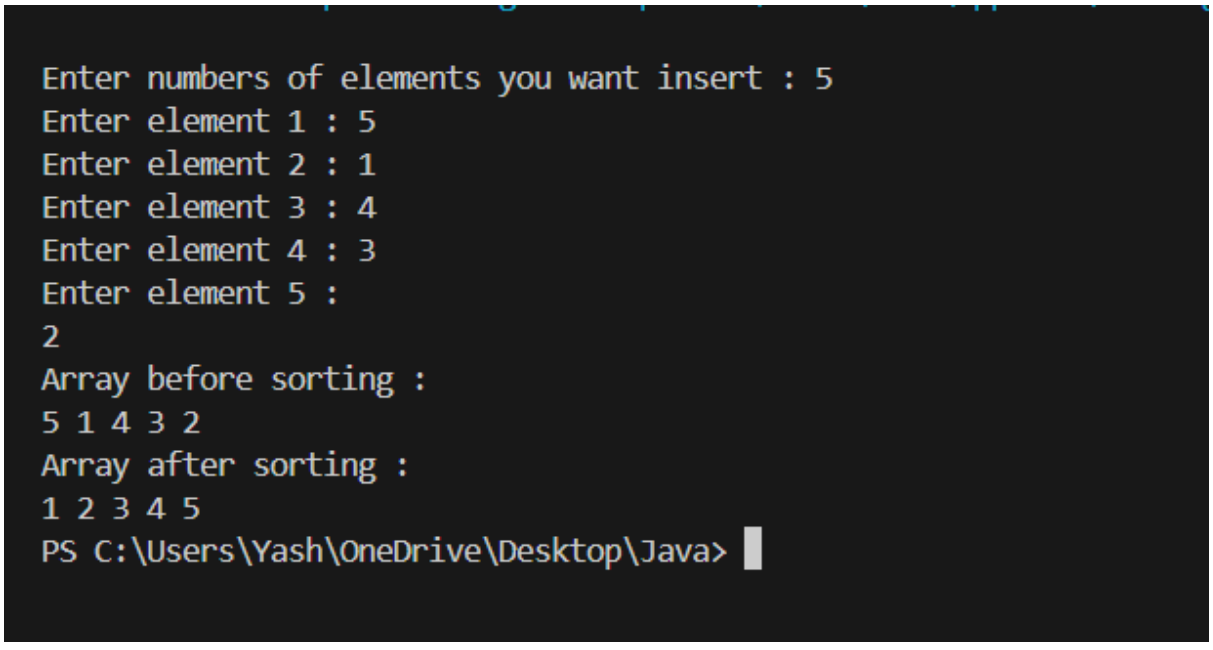
Sorting Techniques:

Buble Sort:-

```
import java.util.*;
public class bubbleSort {
   //print arr
   public static void  display(int[] arr){
      for(int i=0;i<arr.length;i++){
         System.out.print(arr[i]+" ");
      }
   }

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

      System.out.print("Enter numbers of elements you want insert : ");
      int n = sc.nextInt();

      int[] arr = new int[n];

      for (int i=0 ; i<n ; i++) {
         System.out.print("Enter element " + (i+1) + " : ");
         arr[i] = sc.nextInt();
      }

      System.out.println("Array before sorting :");
      display(arr);

      //bubble Sort
      for(int i = 0; i <=arr.length ; i++) {
```

```
        for(int j=0 ; j<arr.length-i-1 ; j++) {
            if(arr[j] > arr[j+1]) {
                //swap
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp ;
            }
        }
    }
    System.out.println("\nArray after sorting :");
    display(arr);
  }
}
```

Output:-



Insertion Sort:-

```
import java.util.*;


public class insertionSort {
   //accpeting array
   public static void getArray(int arr[] , int n) {

      Scanner sc = new Scanner(System.in);


      for(int i=0 ; i<n ; i++){
         System.out.print("Enter "+(i+1)+" number : ");
         arr[i] = sc.nextInt();
      }
   }

   //printing array
   public static void printArray(int arr[] , int n) {
      for(int i=1 ; i<n ; i++) {
         System.out.print(arr[i]+" ");
      }
      System.out.println();
   }

   //Selection sort
   public static void insertionSorting(int arr[] , int n) {
      for(int i=0 ; i<n-1 ; i++) {
         int current = arr[i];
         int j = i-1;
```

```
        while(j>=0 && current<arr[j]) {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = current;
    }
  }

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

    System.out.print("How many elements you want to enter : ");
    int n = sc.nextInt();

    int[] arr = new int[n];
    getArray(arr , n);

    System.out.println("\nBefore sorting : ");
    printArray(arr,n);

    System.out.println("\nAfter Sorting : ");
    insertionSorting(arr , n);
    printArray(arr,n);
  }

}
```

Output:-

```
PS C:\Users\Yash\OneDrive\Desktop\Java>  & 'C:\Program Files\Java\j
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\AppData\Roaming
rt'
How many elements you want to enter : 5
Enter 1 number : 4
Enter 2 number : 2
Enter 3 number : 5
Enter 4 number : 3
Enter 5 number : 1

Before sorting :
2 5 3 1

After Sorting :
3 4 5 1
PS C:\Users\Yash\OneDrive\Desktop\Java>
```

Selection Sort:-

```
import java.util.*;

public class selectionSort {

   //accpeting array
   public static void getArray(int arr[] , int n) {

      Scanner sc = new Scanner(System.in);

      for(int i=0 ; i<n ; i++){
         System.out.print("Enter "+(i+1)+" number : ");
         arr[i] = sc.nextInt();
      }
   }

   //printing array
   public static void printArray(int arr[] , int n) {
      for(int i=0 ; i<n ; i++) {
         System.out.print(arr[i]+" ");
      }
   }

   //Selection sort
   public static void selectionSorting(int arr[] , int n) {
      for(int i=0 ; i<n-1 ; i++) {
         int smallest = i;
         for(int j=i+1 ; j<n ; j++) {
            if(arr[smallest] > arr[j]){
```

```
                smallest = j;
            }
        }
        int temp = arr[i];
        arr[i] = arr[smallest];
        arr[smallest] = temp;
    }
  }

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

    System.out.print("How many elements you want to enter : ");
    int n = sc.nextInt();

    int[] arr = new int[n];
    getArray(arr , n);

    System.out.println("\nBefore sorting : ");
    printArray(arr,n);

    System.out.println("\nAfter Sorting : ");
    selectionSorting(arr , n);
    printArray(arr,n);
  }
}
```

Output:-

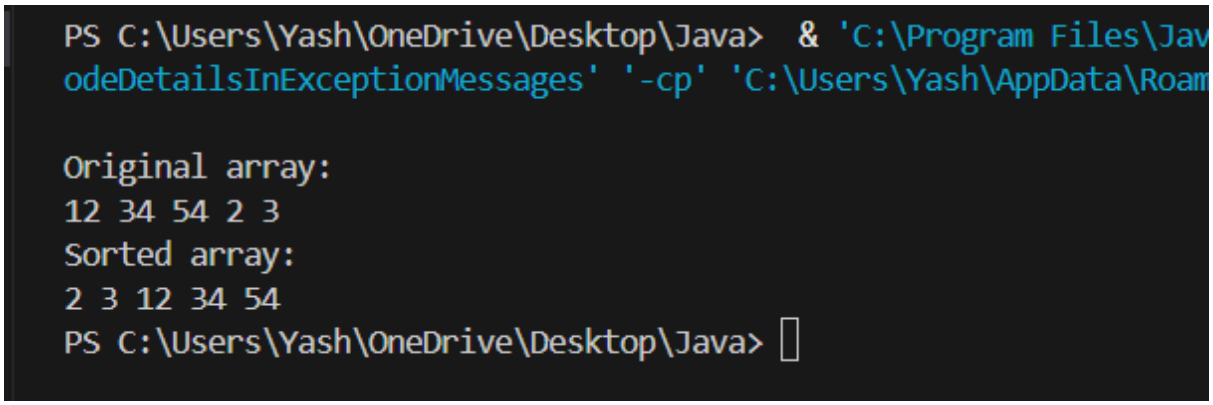
```
PS C:\Users\Yash\OneDrive\Desktop\Java>  & 'C:\Program Files\Java\j
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\AppData\Roaming
rt'
How many elements you want to enter : 5
Enter 1 number : 2
Enter 2 number : 4
Enter 3 number : 3
Enter 4 number : 1
Enter 5 number : 5

Before sorting :
2 4 3 1 5
After Sorting :
1 2 3 4 5
PS C:\Users\Yash\OneDrive\Desktop\Java>
```

Shell Sort:-

```
public class ShellSort {

    public static void shellSort(int[] array) {
        int n = array.length;

        for (int gap = n / 2; gap > 0; gap /= 2) {
            for (int i = gap; i < n; i++) {
                int temp = array[i];
                int j;
                for (j = i; j >= gap && array[j - gap] > temp; j -= gap) {
                    array[j] = array[j - gap];
                }
                array[j] = temp;
            }
        }
    }

    public static void printArray(int[] array) {
        for (int num : array) {
            System.out.print(num + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int[] array = {12, 34, 54, 2, 3};

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

```
        shellSort(array);

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

Output:-



Sorting Techniques:

Linear Search:-

//Jva program for Linear Search

```
import java.util.Scanner;

public class linearSearch {
    public static int searching(int[] array, int key) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == key) {
                return i;
            }
        }
        return -1;
    }

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

        // Input array size
        System.out.print("\nEnter the number of elements in the array: ");
        int n = sc.nextInt();
        int[] array = new int[n];

        // Input array elements
        System.out.println("\nEnter the elements of the array:");
        for (int i = 0; i < n; i++) {
            array[i] = sc.nextInt();
        }
```

```
        // Input the key to search
        System.out.print("\nEnter the element to search for: ");
        int key = sc.nextInt();

        // Perform linear search
        int result = searching(array, key);

        // Display the result
        if (result == -1) {
            System.out.println("\nElement not found in the array.");
        } else {
            System.out.println("\nElement found at index: " + result);
        }
    }
}
```

Output:-

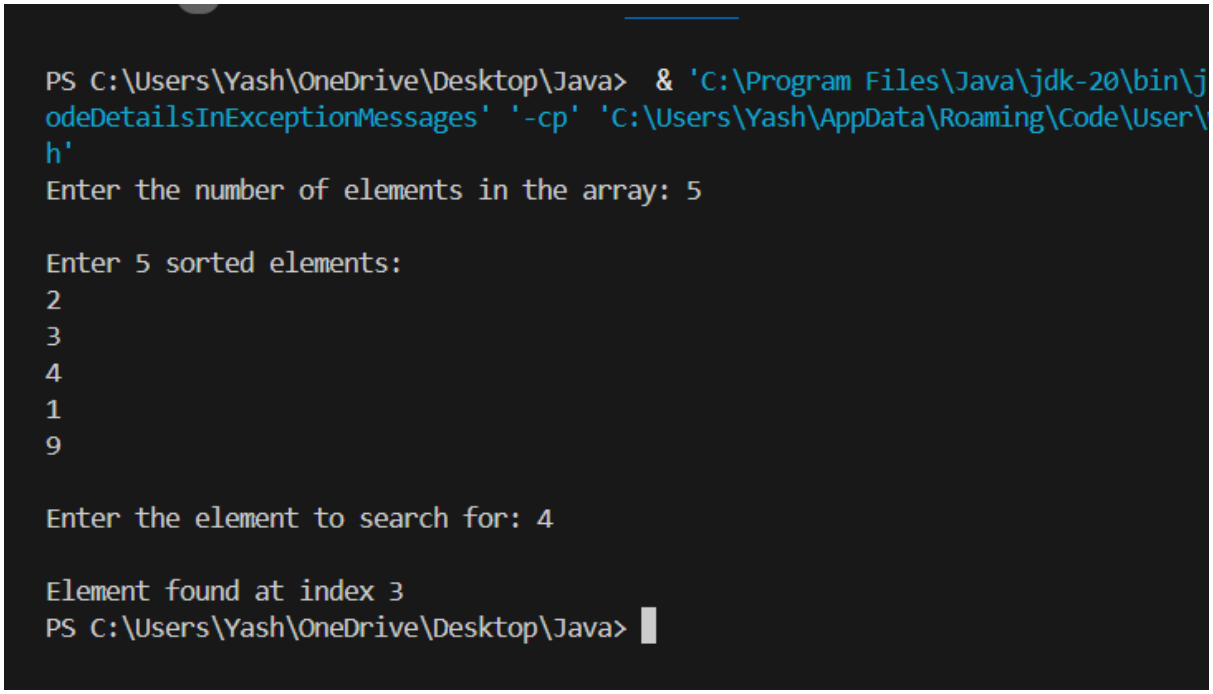
```
PS C:\Users\Yash\OneDrive\Desktop\Java>  & 'C:\Program Files\Java\jdk-20\bin\ja
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\AppData\Roaming\Code\User\w
h'

Enter the number of elements in the array: 5

Enter the elements of the array:
9
5
1
3
7

Enter the element to search for: 5

Element found at index: 1
PS C:\Users\Yash\OneDrive\Desktop\Java>
```

Binary Search:-

```
import java.util.*;

public class binarySearch {
    public static int searching(int[] arr, int target) {
        int left = 0;
        int right = arr.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (arr[mid] == target) {
                return mid;
            }

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

        return -1;
    }

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

        // Input array size
```

```
        System.out.print("Enter the number of elements in the array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];

        // Input array elements
        System.out.println("\nEnter " + n + " sorted elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        // Input target element to search for
        System.out.print("\nEnter the element to search for: ");
        int target = sc.nextInt();

        // Sort array (optional if array is guaranteed sorted by user)
        Arrays.sort(arr);

        // Perform binary search
        int result = searching(arr, target);

        // Output the result
        if (result == -1) {
            System.out.println("\nElement not present in array");
        } else {
            System.out.println("\nElement found at index " + result);
        }
    }
}
```

Output:-


PS C:\Users\Yash\OneDrive\Desktop\Java> & 'C:\Program Files\Java\jdk-20\bin\j
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\AppData\Roaming\Code\User\
h'
Enter the number of elements in the array: 5
Enter 5 sorted elements:
2
3
4
1
9
Enter the element to search for: 4
Element found at index 3
PS C:\Users\Yash\OneDrive\Desktop\Java>

Methods of hashing

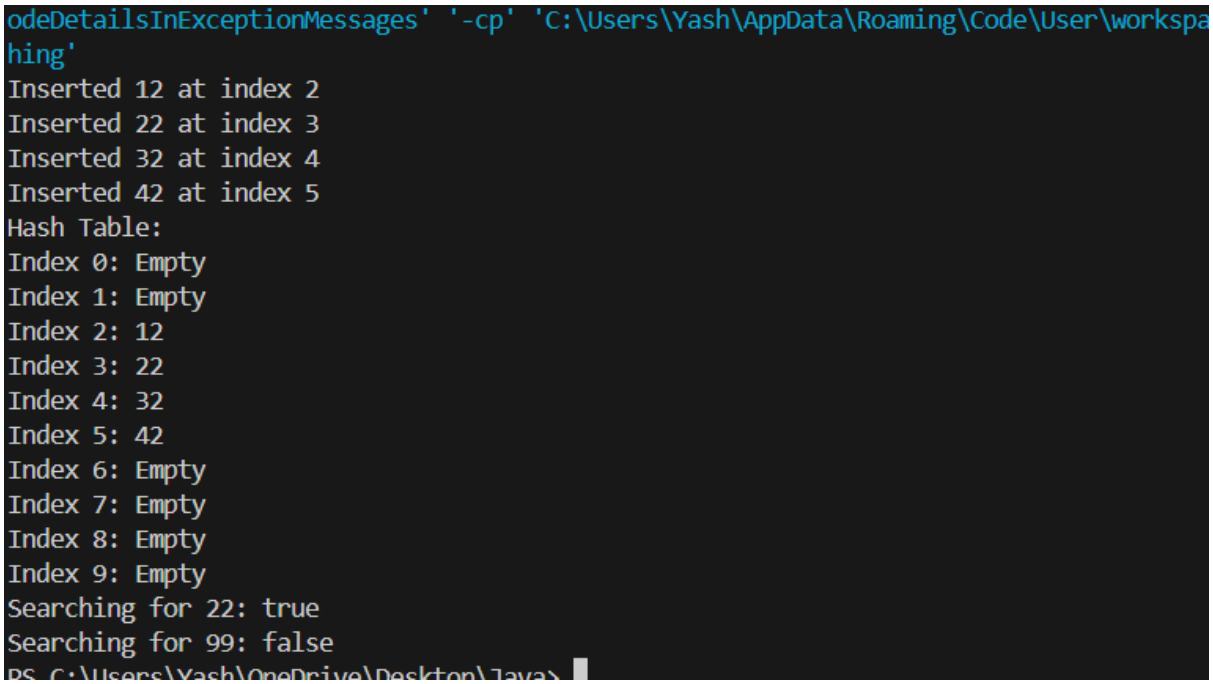
Module Division:

```
public class DivisionHashing {
    private static final int TABLE_SIZE = 10;
    private int[] hashTable;

    public DivisionHashing() {
        hashTable = new int[TABLE_SIZE];
        for (int i = 0; i < TABLE_SIZE; i++) {
            hashTable[i] = -1;
        }
    }

    public int hash(int key) {
        return key % TABLE_SIZE;
    }
    public void insert(int key) {
        int hashValue = hash(key);

        int index = hashValue;
        while (hashTable[index] != -1) {
            index = (index + 1) % TABLE_SIZE;
        }

        hashTable[index] = key;
        System.out.println("Inserted " + key + " at index " + index);
    }

    public boolean search(int key) {
```

```
    int hashValue = hash(key);
    int index = hashValue;

    while (hashTable[index] != -1) {
        if (hashTable[index] == key) {
            return true;
        }
        index = (index + 1) % TABLE_SIZE;
    }
    return false;
}

public void display() {
    System.out.println("Hash Table:");
    for (int i = 0; i < TABLE_SIZE; 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) {
    DivisionHashing hashing = new DivisionHashing();

    // Insert keys
    hashing.insert(12);
    hashing.insert(22);
```

```
        hashing.insert(32);
        hashing.insert(42);

        // Display the hash table
        hashing.display();

        // Search for a key
        System.out.println("Searching for 22: " + hashing.search(22));
        System.out.println("Searching for 99: " + hashing.search(99));
    }
}
```

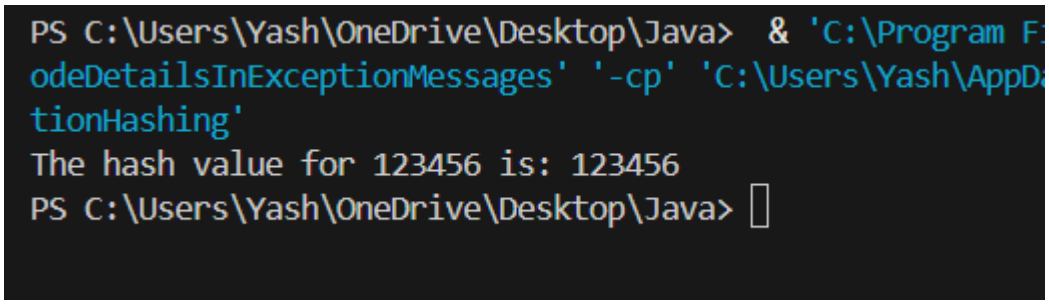
Output:-



Digit Extraction

```
public class DigitExtractionHashing {

    public static int digitExtractionHash(int number) {
        int hash = 0;
        int multiplier = 1;

        while (number > 0) {
            int digit = number % 10;
            hash += digit * multiplier;
            multiplier *= 10;
            number /= 10;
        }

        return hash;
    }

    public static void main(String[] args) {
        int number = 123456;
        int hashValue = digitExtractionHash(number);

        System.out.println("The hash value for " + number + " is: " + hashValue);
    }
}
```

Output:-


PS C:\Users\Yash\OneDrive\Desktop\Java> & 'C:\Program F
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\AppD
tionHashing'
The hash value for 123456 is: 123456
PS C:\Users\Yash\OneDrive\Desktop\Java>

Stack

Array implementaion of Ordinary Queue

```
class Stack {
    private int[] stack;
    private int top;
    private int capacity;

    // Constructor to initialize the stack
    public Stack(int size) {
        stack = new int[size];
        capacity = size;
        top = -1;
    }

    // Add an element to the stack
    public void push(int value) {
        if (isFull()) {
            System.out.println("Stack Overflow: Unable to add " + value);
            return;
        }
        stack[++top] = value;
    }

    // Remove and return the top element of the stack
    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack Underflow: No element to remove");
            return -1;
        }
        return stack[top--];
```

```
}

// Return the top element without removing it
public int peek() {
    if (isEmpty()) {
        System.out.println("Stack is empty");
        return -1;
    }
    return stack[top];
}

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

// Check if the stack is full
public boolean isFull() {
    return top == capacity - 1;
}

// Return the current size of the stack
public int size() {
    return top + 1;
}

// Print the stack elements
public void printStack() {
    if (isEmpty()) {
        System.out.println("Stack is empty");
        return;
```

```
        }
        System.out.print("Stack elements: ");
        for (int i = 0; i <= top; i++) {
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Stack stack = new Stack(5);

        stack.push(10);
        stack.push(20);
        stack.push(30);

        stack.printStack(); // Output: Stack elements: 10 20 30

        System.out.println("Top element: " + stack.peek()); // Output: 30

        System.out.println("Popped element: " + stack.pop()); // Output: 30

        stack.printStack(); // Output: Stack elements: 10 20
    }
}
```

Output:-

```
PS C:\Users\Yash\OneDrive\Desktop\Java>  &
odeDetailsInExceptionMessages' '-cp' 'C:\Us
Stack elements: 10 20 30
Top element: 30
Popped element: 30
Stack elements: 10 20
PS C:\Users\Yash\OneDrive\Desktop\Java>
```

Array implementaion of Queue

```
class Queue {
    private int[] arr;
    private int front;
    private int rear;
    private int capacity;
    private int size;

    public Queue(int capacity) {
        this.capacity = capacity;
        arr = new int[capacity];
        front = 0;
        rear = -1;
        size = 0;
    }

    // Method to add an element to the queue
    public void enqueue(int item) {
        if (isFull()) {
            System.out.println("Queue is full. Cannot enqueue " + item);
            return;
        }
        rear = (rear + 1) % capacity; // Circular increment
        arr[rear] = item;
        size++;
    }

    // Method to remove an element from the queue
    public int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty. Cannot dequeue.");
```

```
        return -1;
    }
    int item = arr[front];
    front = (front + 1) % capacity; // Circular increment
    size--;
    return item;
}

// Method to check if the queue is empty
public boolean isEmpty() {
    return size == 0;
}

// Method to check if the queue is full
public boolean isFull() {
    return size == capacity;
}

// Method to get the size of the queue
public int getSize() {
    return size;
}

// Method to display the elements of the queue
public void display() {
    if (isEmpty()) {
        System.out.println("Queue is empty.");
        return;
    }
    System.out.println("Queue elements: ");
    for (int i = 0; i < size; i++) {
```

```
            System.out.print(arr[(front + i) % capacity] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Queue queue = new Queue(5);

        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        queue.enqueue(40);
        queue.enqueue(50);
        queue.display(); // Displays all elements in the queue

        System.out.println("\nAfter deleting elements from queue :");
        queue.dequeue();
        queue.dequeue();
        queue.display(); // Displays updated queue

        System.out.println("\nAfter adding elements from queue :");
        queue.enqueue(60);
        queue.enqueue(70);
        queue.display(); // Displays updated queue
    }
}
```

Output:-

```
PS C:\Users\Yash\OneDrive\Desktop\Java>  & 'C:\Program Files\Java\jd
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\AppData\Roaming'
Queue elements:
10 20 30 40 50

After deleting elements from queue :
Queue elements:
30 40 50

After adding elements from queue :
Queue elements:
30 40 50 60 70
PS C:\Users\Yash\OneDrive\Desktop\Java>
```

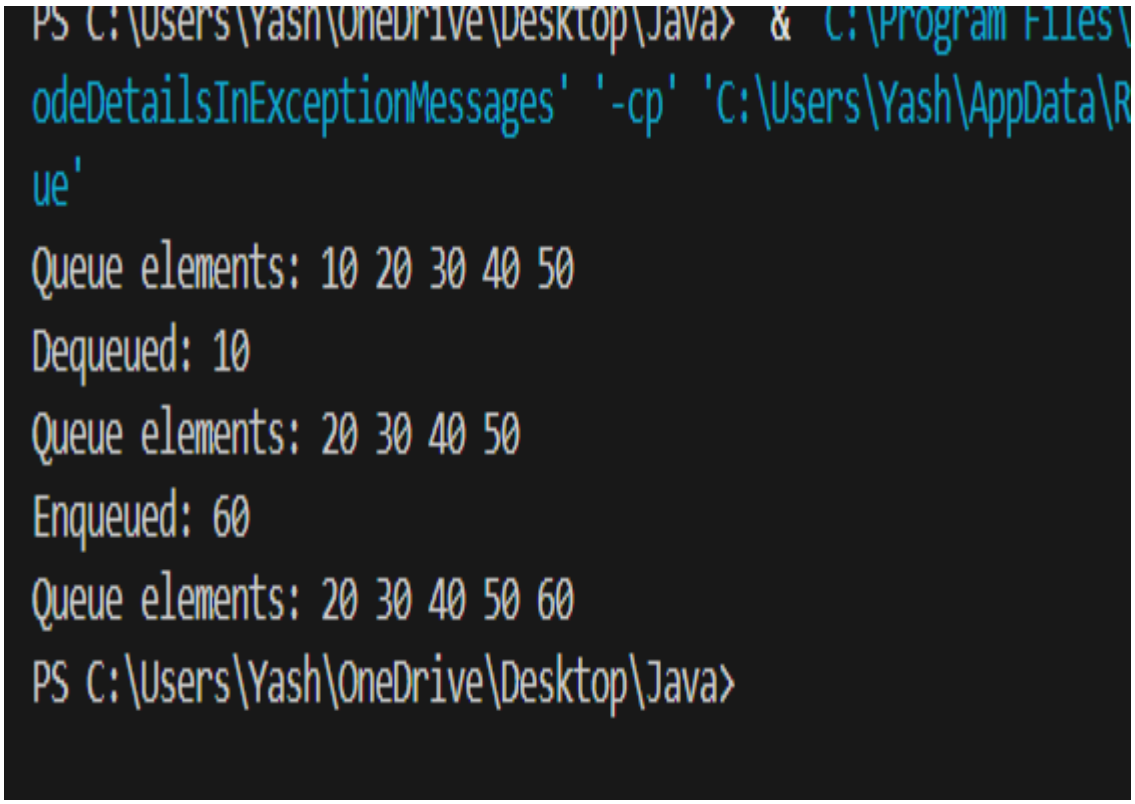
Array implementaion of Circuler Queue

```
class CircularQueue {
    private int[] queue;
    private int front, rear, size, capacity;

    public CircularQueue(int capacity) {
        this.capacity = capacity;
        queue = new int[capacity];
        front = rear = -1;
        size = 0;
    }

    public boolean isFull() {
        return size == capacity;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public void enqueue(int value) {
        if (isFull()) {
            System.out.println("Queue is full! Cannot enqueue.");
            return;
        }
        if (front == -1) {
            front = 0;
        }
        rear = (rear + 1) % capacity;
```

```
    queue[rear] = value;
    size++;
}

public int dequeue() {
    if (isEmpty()) {
        System.out.println("Queue is empty! Cannot dequeue.");
        return -1;
    }
    int dequeuedValue = queue[front];
    front = (front + 1) % capacity;
    size--;
    if (size == 0) {
        front = rear = -1;
    }
    return dequeuedValue;
}

public int front() {
    if (isEmpty()) {
        System.out.println("Queue is empty!");
        return -1;
    }
    return queue[front];
}

public void printQueue() {
    if (isEmpty()) {
        System.out.println("Queue is empty!");
```

```
            return;
        }
        System.out.print("Queue elements: ");
        for (int i = 0; i < size; i++) {
            System.out.print(queue[(front + i) % capacity] + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        CircularQueue cq = new CircularQueue(5);

        cq.enqueue(10);
        cq.enqueue(20);
        cq.enqueue(30);
        cq.enqueue(40);
        cq.enqueue(50);

        cq.printQueue();

        System.out.println("Dequeued: " + cq.dequeue());
        cq.printQueue();

        cq.enqueue(60);
        System.out.println("Enqueued: 60");

        cq.printQueue();
    }
}
```

Ouput:-


PS C:\Users\Yash\OneDrive\Desktop\Java> & 'C:\Program Files\
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\AppData\R
ue'
Queue elements: 10 20 30 40 50
Dequeued: 10
Queue elements: 20 30 40 50
Enqueued: 60
Queue elements: 20 30 40 50 60
PS C:\Users\Yash\OneDrive\Desktop\Java>

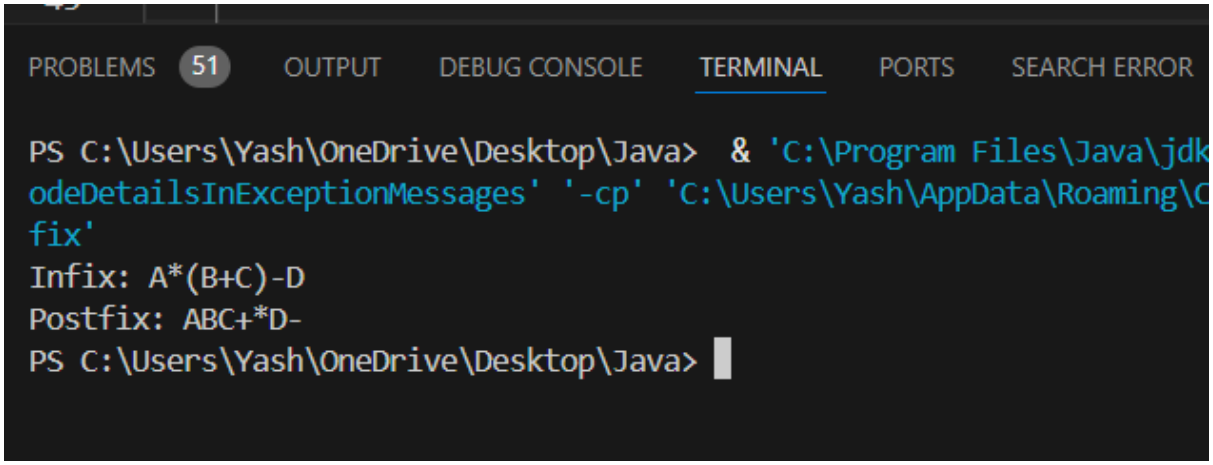
Conversion of Infix notation to Postfix Notation

```
import java.util.Stack;


public class InfixToPostfix {
    static int precedence(char ch) {
        switch (ch) {
            case '+':
            case '-':
                return 1;
            case '*':
            case '/':
                return 2;
            case '^':
                return 3;
            default:
                return -1;
        }
    }

    public static String infixToPostfix(String infix) {
        Stack<Character> stack = new Stack<>();
        StringBuilder postfix = new StringBuilder();

        for (int i = 0; i < infix.length(); i++) {
            char currentChar = infix.charAt(i);

            if (Character.isLetterOrDigit(currentChar)) {
                postfix.append(currentChar);
            }
```

```
        else if (currentChar == '(') {
            stack.push(currentChar);
        }
        else if (currentChar == ')') {
            while (!stack.isEmpty() && stack.peek() != '(') {
                postfix.append(stack.pop());
            }
            stack.pop();
        }
        else {
            while (!stack.isEmpty() && precedence(stack.peek()) >= precedence(currentChar)) {
                postfix.append(stack.pop());
            }
            stack.push(currentChar);
        }
    }

    while (!stack.isEmpty()) {
        postfix.append(stack.pop());
    }

    return postfix.toString();
}

public static void main(String[] args) {
    String infixExpression = "A*(B+C)-D";
    String postfixExpression = infixToPostfix(infixExpression);
    System.out.println("Infix: " + infixExpression);
    System.out.println("Postfix: " + postfixExpression);
```

```
    }
}
```

Output:-

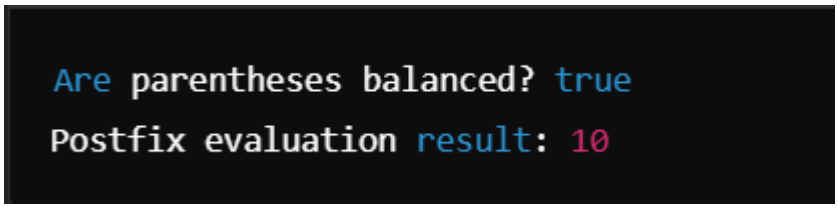

PROBLEMS 51
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
SEARCH ERROR
PS C:\Users\Yash\OneDrive\Desktop\Java> & 'C:\Program Files\Java\jdk
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\AppData\Roaming\C
fix'
Infix: A*(B+C)-D
Postfix: ABC+*D-
PS C:\Users\Yash\OneDrive\Desktop\Java>

```
import java.util.Stack;

public class PostfixEvaluationAndParenthesesBalance {

    public static boolean areParenthesesBalanced(String expression) {
        Stack<Character> stack = new Stack<>();

        for (char ch : expression.toCharArray()) {
            if (ch == '(') {
                stack.push(ch);
            } else if (ch == ')') {
                if (stack.isEmpty()) {
                    return false;
                }
                stack.pop();
            }
        }

        return stack.isEmpty();
    }

    public static int evaluatePostfix(String expression) {
        Stack<Integer> stack = new Stack<>();

        for (char ch : expression.toCharArray()) {
            if (Character.isDigit(ch)) {
                stack.push(ch - '0');
            }
            else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
```

```
            int operand2 = stack.pop();
            int operand1 = stack.pop();
            int result = 0;

            switch (ch) {
               case '+':
                  result = operand1 + operand2;
                  break;
               case '-':
                  result = operand1 - operand2;
                  break;
               case '*':
                  result = operand1 * operand2;
                  break;
               case '/':
                  result = operand1 / operand2;
                  break;
            }
            stack.push(result);
         }
      }

      return stack.pop();
   }

   public static void main(String[] args) {
      String expression = "(2 + 3) * (4 / (1 + 1))";
      String postfix = "23+41+/*";
```

```
        System.out.println("Are parentheses balanced? " +
areParenthesesBalanced(expression));


        System.out.println("Postfix evaluation result: " + evaluatePostfix(postfix));
    }
}
```

Output:-



Linked List

Singly Linked List(Insert , Display , Delete , Delete , Search , Count , Reverse)

```
public class SinglyLinkedList {

   class Node {
      int data;
      Node next;

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

   private Node head;

   public SinglyLinkedList() {
      head = null;
   }

   public void insert(int data) {
      Node newNode = new Node(data);
      if (head == null) {
         head = newNode;
      } else {
         Node temp = head;
         while (temp.next != null) {
            temp = temp.next;
         }
```

```
            temp.next = newNode;
        }
    }

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

    public void delete(int data) {
        if (head == null) {
            System.out.println("List is empty.");
            return;
        }
        if (head.data == data) {
            head = head.next;
            return;
        }

        Node temp = head;
        while (temp.next != null && temp.next.data != data) {
```

```
            temp = temp.next;
        }
        if (temp.next == null) {
            System.out.println("Element not found.");
        } else {
            temp.next = temp.next.next;
        }
    }

    public boolean search(int data) {
        Node temp = head;
        while (temp != null) {
            if (temp.data == data) {
                return true;
            }
            temp = temp.next;
        }
        return false;
    }

    public int count() {
        int count = 0;
        Node temp = head;
        while (temp != null) {
            count++;
            temp = temp.next;
        }
        return count;
    }
```

```
public void reverse() {
    Node prev = null;
    Node current = head;
    Node next = null;
    while (current != null) {
        next = current.next;
        current.next = prev;
        prev = current;
        current = next;
    }
    head = prev;
}

public static void main(String[] args) {
    SinglyLinkedList list = new SinglyLinkedList();

    list.insert(10);
    list.insert(20);
    list.insert(30);
    list.insert(40);

    System.out.println("List after insertion:");
    list.display();

    System.out.println("Count of nodes: " + list.count());

    System.out.println("Search for 20: " + (list.search(20) ? "Found" : "Not Found"));
```

```
        list.delete(20);
        System.out.println("List after deleting 20:");
        list.display();

        list.reverse();
        System.out.println("List after reversing:");
        list.display();
    }
}
```

Output:-

```
PS C:\Users\Yash\OneDrive\Desktop\Java>  & 'C:\Program Files\Java\jdk
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\AppData\Roaming\C
dList'
List after insertion:
10 20 30 40
Count of nodes: 4
Search for 20: Found
List after deleting 20:
10 30 40
List after reversing:
40 30 10
PS C:\Users\Yash\OneDrive\Desktop\Java>
```

Doubly Linked List(Insert , Display , Delete , Delete , Search , Count , Reverse)

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

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

    private Node head, tail;

    public DoublyLinkedList() {
        head = tail = null;
    }

    public void insert(int data) {
        Node newNode = new Node(data);

        if (head == null) {
            head = tail = newNode;
        } else {
            tail.next = newNode;
            newNode.prev = tail;
            tail = newNode;
        }
    }
```

```
public void display() {
    if (head == null) {
        System.out.println("The list is empty.");
        return;
    }

    Node current = head;
    while (current != null) {
        System.out.print(current.data + " ");
        current = current.next;
    }
}

public void delete(int data) {
    if (head == null) {
        System.out.println("The list is empty.");
        return;
    }

    Node current = head;

    if (current.data == data) {
        head = current.next;
        if (head != null) {
            head.prev = null;
        }
        return;
    }
```

```
    while (current != null && current.data != data) {
        current = current.next;
    }

    if (current == null) {
        System.out.println("Node with data " + data + " not found.");
        return;
    }

    if (current.next == null) {
        current.prev.next = null;
        tail = current.prev;
    } else {
        current.prev.next = current.next;
        current.next.prev = current.prev;
    }
}

public boolean search(int data) {
    Node current = head;
    while (current != null) {
        if (current.data == data) {
            return true;
        }
        current = current.next;
    }
    return false;
}
```

Circular Linked List(Insert , Display , Delete , Delete , Search , Count , Reverse)

```
  public int count() {
    int count = 0;
    Node current = head;
    while (current != null) {
      count++;
      current = current.next;
    }
    return count;
  }

  public static void main(String[] args) {
    DoublyLinkedList list = new DoublyLinkedList();

    list.insert(10);
    list.insert(20);
    list.insert(30);
    list.insert(40);

    System.out.println("List contents:");
    list.display();

    System.out.println("\nSearching for 20: " + list.search(20));
    System.out.println("Searching for 50: " + list.search(50));

    System.out.println("Number of nodes: " + list.count());

    list.delete(20);
    System.out.println("List after deleting 20:");
```

```
        list.display();

        list.delete(50);

        System.out.println("List after deletion:");
        list.display();
    }
}
```

Output:-

```
PS C:\Users\Yash\OneDrive\Desktop\Java>  & 'C:\Program Files\Java\jdk-20\bin\java.exe
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\AppData\Roaming\Code\User\workspac
dList'
List contents:
10 20 30 40
Searching for 20: true
Searching for 50: false
Number of nodes: 4
List after deleting 20:
10 30 40 Node with data 50 not found.
List after deletion:
10 30 40
PS C:\Users\Yash\OneDrive\Desktop\Java>
```

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

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

    private Node last;

    public CircularLinkedList() {
        last = null;
    }

    public void insert(int data) {
        Node newNode = new Node(data);
        if (last == null) {
            last = newNode;
            last.next = last;
        } else {
            newNode.next = last.next;
            last.next = newNode;
            last = newNode;
        }
```

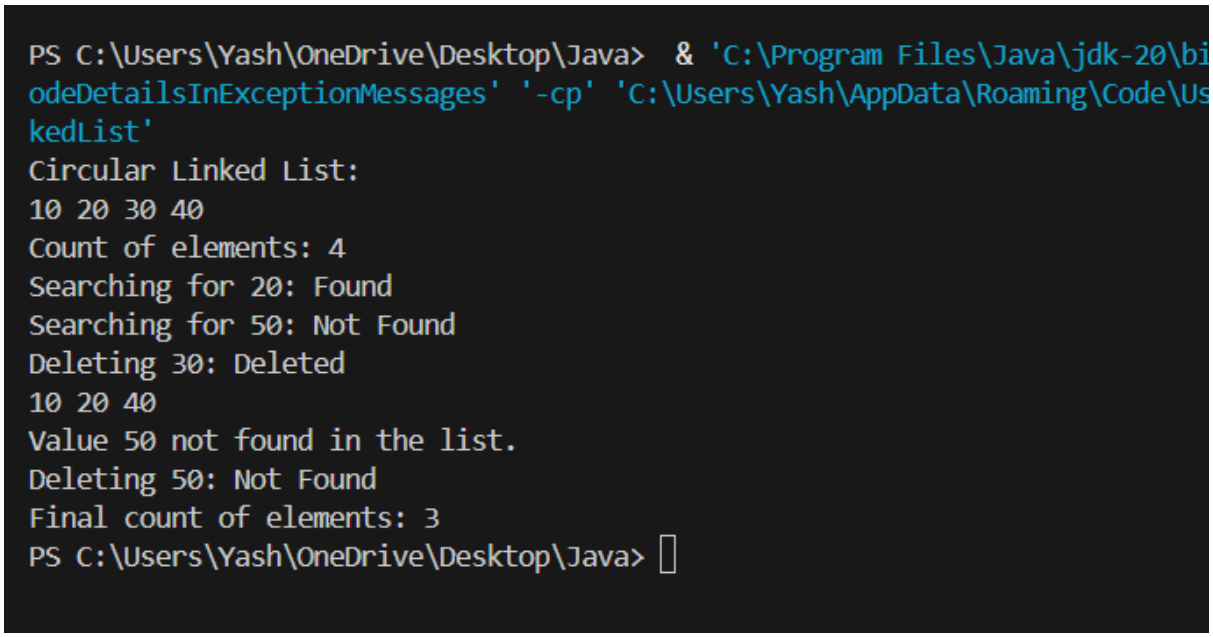
```
}

public void display() {
    if (last == null) {
        System.out.println("List is empty.");
        return;
    }
    Node temp = last.next;
    do {
        System.out.print(temp.data + " ");
        temp = temp.next;
    } while (temp != last.next);
    System.out.println();
}

public boolean delete(int value) {
    if (last == null) {
        System.out.println("List is empty. Cannot delete.");
        return false;
    }

    Node current = last.next, previous = last;

    do {
        if (current.data == value) {
            if (current == last && current.next == last) {
                last = null;
            } else if (current == last) {
                previous.next = last.next;
```

```
                last = previous;
            } else {
                previous.next = current.next;
            }
            return true;
        }
        previous = current;
        current = current.next;
    } while (current != last.next);

    System.out.println("Value " + value + " not found in the list.");
    return false;
}

public boolean search(int value) {
    if (last == null) {
        return false;
    }

    Node temp = last.next;
    do {
        if (temp.data == value) {
            return true;
        }
        temp = temp.next;
    } while (temp != last.next);

    return false;
}
```

```
public int count() {
    if (last == null) {
        return 0;
    }

    int count = 0;
    Node temp = last.next;
    do {
        count++;
        temp = temp.next;
    } while (temp != last.next);

    return count;
}

public static void main(String[] args) {
    CircularLinkedList cll = new CircularLinkedList();

    cll.insert(10);
    cll.insert(20);
    cll.insert(30);
    cll.insert(40);

    System.out.println("Circular Linked List:");
    cll.display();

    System.out.println("Count of elements: " + cll.count());
```

```
        System.out.println("Searching for 20: " + (cll.search(20) ? "Found" : "Not Found"));
        System.out.println("Searching for 50: " + (cll.search(50) ? "Found" : "Not Found"));

        System.out.println("Deleting 30: " + (cll.delete(30) ? "Deleted" : "Not Found"));
        cll.display();

        System.out.println("Deleting 50: " + (cll.delete(50) ? "Deleted" : "Not Found"));

        System.out.println("Final count of elements: " + cll.count());
    }
}
```

Output:-



Polynomial Addition using Linked list

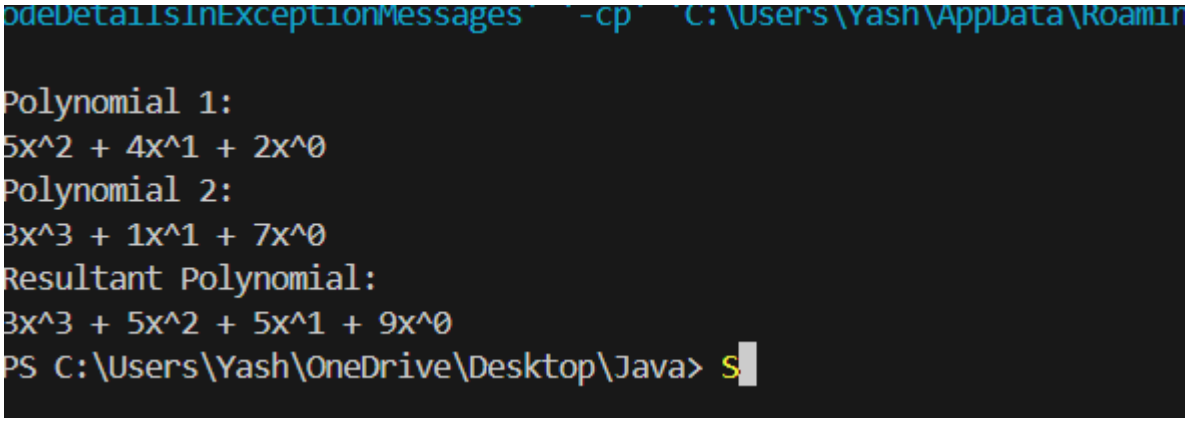
```
class Polynomial {
    static class Node {
        int coefficient;
        int exponent;
        Node next;

        Node(int coefficient, int exponent) {
            this.coefficient = coefficient;
            this.exponent = exponent;
            this.next = null;
        }
    }

    Node head;

    // Add a new term to the polynomial
    public void addTerm(int coefficient, int exponent) {
        Node newNode = new Node(coefficient, exponent);
        if (head == null || head.exponent < exponent) {
            newNode.next = head;
            head = newNode;
        } else {
            Node current = head;
            while (current.next != null && current.next.exponent >= exponent) {
                current = current.next;
            }

            if (current.exponent == exponent) {
```

```
            current.coefficient += coefficient;
        } else {
            newNode.next = current.next;
            current.next = newNode;
        }
    }
}

// Add two polynomials
public static Polynomial add(Polynomial poly1, Polynomial poly2) {
    Polynomial result = new Polynomial();
    Node p1 = poly1.head;
    Node p2 = poly2.head;

    while (p1 != null && p2 != null) {
        if (p1.exponent == p2.exponent) {
            result.addTerm(p1.coefficient + p2.coefficient, p1.exponent);
            p1 = p1.next;
            p2 = p2.next;
        } else if (p1.exponent > p2.exponent) {
            result.addTerm(p1.coefficient, p1.exponent);
            p1 = p1.next;
        } else {
            result.addTerm(p2.coefficient, p2.exponent);
            p2 = p2.next;
        }
    }

    while (p1 != null) {
```

```
            result.addTerm(p1.coefficient, p1.exponent);
            p1 = p1.next;
        }

        while (p2 != null) {
            result.addTerm(p2.coefficient, p2.exponent);
            p2 = p2.next;
        }

        return result;
    }

    // Display the polynomial
    public void display() {
        Node current = head;
        while (current != null) {
            System.out.print(current.coefficient + "x^" + current.exponent);
            if (current.next != null) {
                System.out.print(" + ");
            }
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Polynomial poly1 = new Polynomial();
        poly1.addTerm(5, 2);
        poly1.addTerm(4, 1);
```

```
        poly1.addTerm(2, 0);

        Polynomial poly2 = new Polynomial();
        poly2.addTerm(3, 3);
        poly2.addTerm(1, 1);
        poly2.addTerm(7, 0);

        System.out.println("Polynomial 1:");
        poly1.display();

        System.out.println("Polynomial 2:");
        poly2.display();

        Polynomial result = add(poly1, poly2);
        System.out.println("Resultant Polynomial:");
        result.display();
    }
}
```

Output:-



Linked List implementation of stack, ordinary queue

i.In Stack:-

```
class Node {
   int data;
   Node next;

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

class Stack {
   private Node top;

   public Stack() {
      this.top = null;
   }

   public void push(int data) {
      Node newNode = new Node(data);
      newNode.next = top;
      top = newNode;
   }

   public int pop() {
      if (isEmpty()) {
         throw new RuntimeException("Stack underflow!");
      }
      int data = top.data;
      top = top.next;
      return data;
   }

   public int peek() {
      if (isEmpty()) {
         throw new RuntimeException("Stack is empty!");
      }
      return top.data;
   }

   public boolean isEmpty() {
      return top == null;
   }
```

```
    public void display() {
        Node current = top;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
    }
}

public class LinkedListStack {
    public static void main(String[] args) {
        Stack stack = new Stack();
        stack.push(10);
        stack.push(20);
        stack.push(30);

        System.out.println("Eements:-");
        stack.display();

        System.out.println("\nPopped: " + stack.pop());
        System.out.println("\nAfter deleting :-");
        stack.display();

        System.out.println("\nTop element: " + stack.peek());

        stack.display();
    }
}
```

Output:-

```
PS C:\Users\Yash\OneDrive\Desktop\Java>  & 'C:\Program Files\J
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\AppData\Ro
tack'
Eements:-
30 20 10
Popped: 30

After deleting :-
20 10
Top element: 20
20 10
PS C:\Users\Yash\OneDrive\Desktop\Java>
```

ii.In Queue:-

```
class Node {
    int data;
    Node next;

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

class LinkedListQueue {
    private Node front, rear;

    public LinkedListQueue() {
        this.front = this.rear = null;
    }

    public void enqueue(int data) {
        Node newNode = new Node(data);

        if (rear == null) {
            front = rear = newNode;
            return;
        }

        rear.next = newNode;
        rear = newNode;
    }

    public int dequeue() {
        if (front == null) {
            throw new IllegalStateException("Queue is empty");
        }

        int data = front.data;
        front = front.next;

        if (front == null) {
            rear = null;
        }

        return data;
    }

    public boolean isEmpty() {
        return front == null;
```

```
    }

    public int peek() {
        if (front == null) {
            throw new IllegalStateException("Queue is empty");
        }
        return front.data;
    }

    public void display() {
        if (front == null) {
            System.out.println("Queue is empty");
            return;
        }

        Node current = front;
        System.out.print("Queue elements: ");
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        LinkedListQueue queue = new LinkedListQueue();

        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);

        System.out.println("Front element: " + queue.peek());

        System.out.println("Dequeued: " + queue.dequeue());
        System.out.println("Dequeued: " + queue.dequeue());

        System.out.println("After deleting:-");
        queue.display();

        System.out.println("Is queue empty? " + queue.isEmpty());

        System.out.println("Dequeued: " + queue.dequeue());

        System.out.println("Is queue empty? " + queue.isEmpty());
    }
}
```

Output:-

```
PS C:\Users\Yash\OneDrive\Desktop\Java>  & 'C:\Program Files\Java\jdk-20\bin\
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\AppData\Roaming\Code\User
ueue'
Front element: 10
Dequeued: 10
Dequeued: 20
After deleting:-
Queue elements: 30
Is queue empty? false
Dequeued: 30
Is queue empty? true
PS C:\Users\Yash\OneDrive\Desktop\Java>
```

iii.Priority queue:-

```
class Node {
    int data;
    int priority;
    Node next;

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

class PriorityQueue {
    private Node head;

    public PriorityQueue() {
        head = null;
    }

    // Insert a node into the priority queue
    public void enqueue(int data, int priority) {
        Node newNode = new Node(data, priority);
        // If the list is empty or new node has higher priority than the head
        if (head == null || priority < head.priority) {
            newNode.next = head;
            head = newNode;
        } else {
            // Traverse the list to find the proper position for the new node
```

```
        Node current = head;
        while (current.next != null && current.next.priority <= priority) {
            current = current.next;
        }
        newNode.next = current.next;
        current.next = newNode;
    }
}

// Remove and return the highest-priority node (head of the list)
public int dequeue() {
    if (head == null) {
        throw new IllegalStateException("Priority queue is empty!");
    }
    int value = head.data;
    head = head.next;
    return value;
}

// Peek at the highest-priority element without removing it
public int peek() {
    if (head == null) {
        throw new IllegalStateException("Priority queue is empty!");
    }
    return head.data;
}

// Check if the priority queue is empty
public boolean isEmpty() {
```

```
        return head == null;
    }

    // Print the priority queue for debugging
    public void printQueue() {
        Node current = head;
        while (current != null) {
            System.out.print("(" + current.data + ", " + current.priority + ") ");
            current = current.next;
        }
        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        PriorityQueue pq = new PriorityQueue();

        pq.enqueue(10, 2);
        pq.enqueue(20, 1);
        pq.enqueue(30, 3);

        System.out.println("Priority Queue after enqueuing:");
        pq.printQueue();

        System.out.println("Dequeued: " + pq.dequeue());
        System.out.println("Priority Queue after dequeuing:");
        pq.printQueue();
```

```
        System.out.println("Peek: " + pq.peek());
        System.out.println("Is Empty: " + pq.isEmpty());
    }
}
```

Output:-

```
Priority Queue after enqueuing:
(20, 1) (10, 2) (30, 3)
Dequeued: 20
Priority Queue after dequeuing:
(10, 2) (30, 3)
Peek: 10
Is Empty: false
```

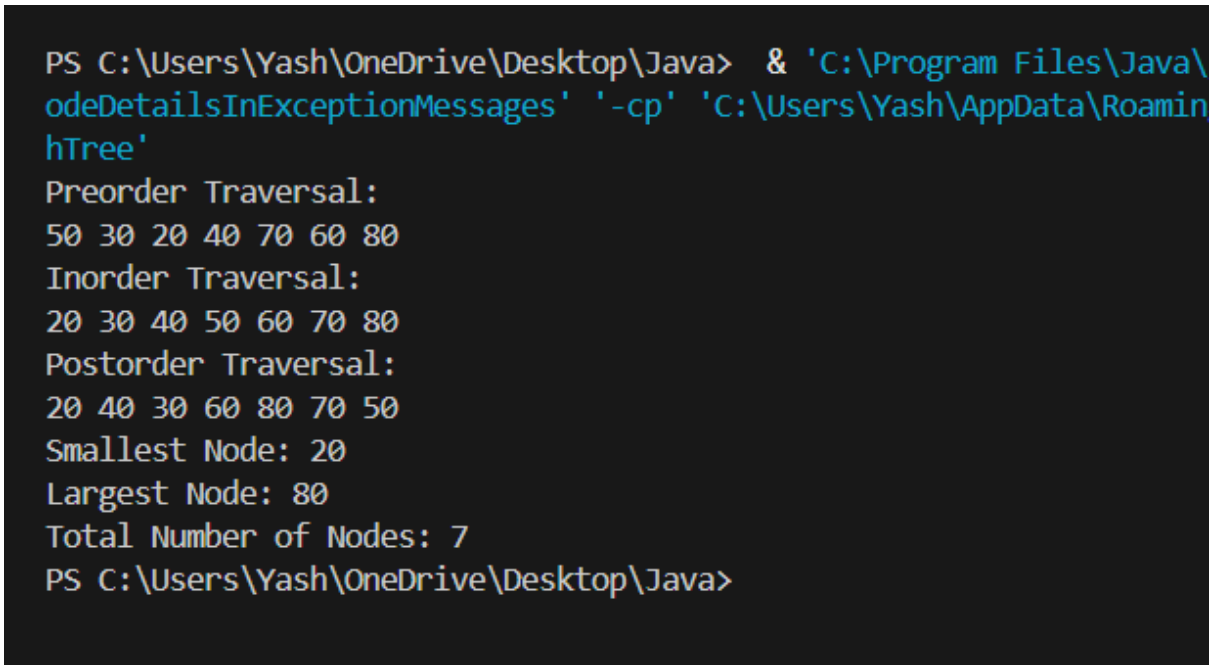
Trees: Binary search tree: Create, Recursive traversal: preorder, postorder, inorder, Search Largest Node, Smallest Node, Count number of nodes

```
class BinarySearchTree {
   static class Node {
      int data;
      Node left, right;

      public Node(int item) {
         data = item;
         left = right = null;
      }
   }

   Node root;

   Node insert(Node root, int data) {
      if (root == null) {
         root = new Node(data);
         return root;
      }
      if (data < root.data)
         root.left = insert(root.left, data);
      else if (data > root.data)
         root.right = insert(root.right, data);
      return root;
   }

   void preorder(Node root) {
      if (root != null) {
         System.out.print(root.data + " ");
         preorder(root.left);
         preorder(root.right);
      }
   }

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

   void postorder(Node root) {
      if (root != null) {
         postorder(root.left);
         postorder(root.right);
```

```
        System.out.print(root.data + " ");
    }
}

int findMin(Node root) {
    while (root.left != null) {
        root = root.left;
    }
    return root.data;
}

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

int countNodes(Node root) {
    if (root == null)
        return 0;
    return 1 + countNodes(root.left) + countNodes(root.right);
}

public static void main(String[] args) {
    BinarySearchTree bst = new BinarySearchTree();
    bst.root = bst.insert(bst.root, 50);
    bst.insert(bst.root, 30);
    bst.insert(bst.root, 70);
    bst.insert(bst.root, 20);
    bst.insert(bst.root, 40);
    bst.insert(bst.root, 60);
    bst.insert(bst.root, 80);

    System.out.println("Preorder Traversal:");
    bst.preorder(bst.root);
    System.out.println();

    System.out.println("Inorder Traversal:");
    bst.inorder(bst.root);
    System.out.println();

    System.out.println("Postorder Traversal:");
    bst.postorder(bst.root);
    System.out.println();

    System.out.println("Smallest Node: " + bst.findMin(bst.root));
```

```
        System.out.println("Largest Node: " + bst.findMax(bst.root));
        System.out.println("Total Number of Nodes: " + bst.countNodes(bst.root));
    }
}
```

Output:-



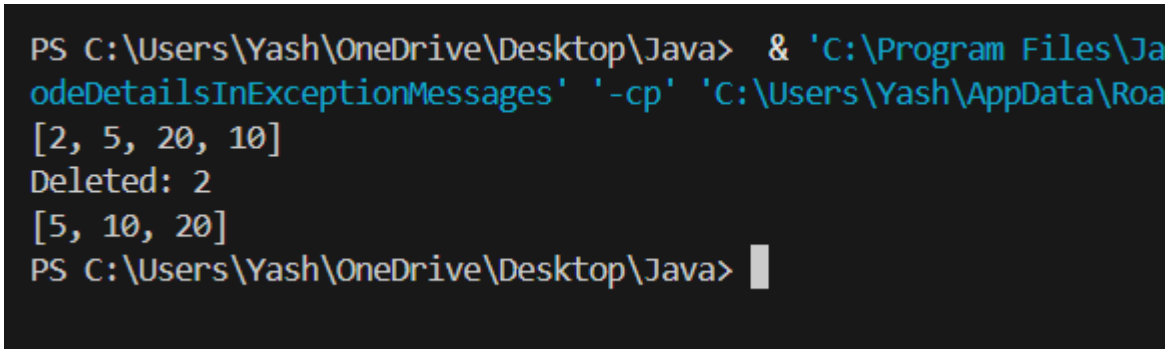
Heap: MinHeap, MaxHeap: reheapUp, reheapDown, Delete.

```
import java.util.ArrayList;


public class MinHeap {
    private ArrayList<Integer> heap;

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

    public void insert(int value) {
        heap.add(value);
        reheapUp(heap.size() - 1);
    }

    public int delete() {
        if (heap.isEmpty()) {
            throw new IllegalStateException("Heap is empty");
        }
        int root = heap.get(0);
        heap.set(0, heap.remove(heap.size() - 1));
        reheapDown(0);
        return root;
    }

    private void reheapUp(int index) {
        int parent = (index - 1) / 2;
        while (index > 0 && heap.get(index) < heap.get(parent)) {
            swap(index, parent);
```

```
        index = parent;
        parent = (index - 1) / 2;
    }
}

private void reheapDown(int index) {
    int leftChild, rightChild, smallest;
    while (index < heap.size()) {
        leftChild = 2 * index + 1;
        rightChild = 2 * index + 2;
        smallest = index;

        if (leftChild < heap.size() && heap.get(leftChild) < heap.get(smallest)) {
            smallest = leftChild;
        }
        if (rightChild < heap.size() && heap.get(rightChild) < heap.get(smallest)) {
            smallest = rightChild;
        }
        if (smallest == index) {
            break;
        }
        swap(index, smallest);
        index = smallest;
    }
}

private void swap(int i, int j) {
    int temp = heap.get(i);
    heap.set(i, heap.get(j));
```

```
        heap.set(j, temp);
    }

    public void printHeap() {
        System.out.println(heap);
    }

    public static void main(String[] args) {
        MinHeap minHeap = new MinHeap();
        minHeap.insert(10);
        minHeap.insert(5);
        minHeap.insert(20);
        minHeap.insert(2);

        minHeap.printHeap();

        System.out.println("Deleted: " + minHeap.delete());
        minHeap.printHeap();
    }
}
```

Output:-



Graphs: Represent a graph using the Adjacency Matrix

```
import java.util.Scanner;

public class Graph {
    private int[][] adjacencyMatrix;
    private int numVertices;

    public Graph(int numVertices) {
        this.numVertices = numVertices;
        adjacencyMatrix = new int[numVertices][numVertices];
    }

    public void addEdge(int source, int destination) {
        adjacencyMatrix[source][destination] = 1;
        adjacencyMatrix[destination][source] = 1;
    }

    public void removeEdge(int source, int destination) {
        adjacencyMatrix[source][destination] = 0;
        adjacencyMatrix[destination][source] = 0;
    }

    public void displayMatrix() {
        for (int i = 0; i < numVertices; i++) {
            for (int j = 0; j < numVertices; j++) {
                System.out.print(adjacencyMatrix[i][j] + " ");
            }
            System.out.println();
        }
```

```
    }

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

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

        Graph graph = new Graph(vertices);

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

        System.out.println("Enter the edges (source and destination):");
        for (int i = 0; i < edges; i++) {
            int source = scanner.nextInt();
            int destination = scanner.nextInt();
            graph.addEdge(source, destination);
        }

        System.out.println("Adjacency Matrix:");
        graph.displayMatrix();

        scanner.close();
    }
}
```

Output:-

```
PS C:\Users\Yash\OneDrive\Desktop\Java>  & 'C:\Program Files\Java\jdk-1
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\AppData\Roaming\Co
Enter the number of vertices: 4
Enter the number of edges: 3
Enter the edges (source and destination):
0 1
1 2
2 3
Adjacency Matrix:
0 1 0 0
1 0 1 0
0 1 0 1
0 0 1 0
PS C:\Users\Yash\OneDrive\Desktop\Java>
```

BFS& DFS on Graph

```
import java.util.*;

class Graph {
    private Map<Integer, List<Integer>> adjList;

    // Constructor
    public Graph() {
        adjList = new HashMap<>();
    }

    // Add an edge to the graph (undirected)
    public void addEdge(int u, int v) {
        adjList.putIfAbsent(u, new ArrayList<>());
        adjList.putIfAbsent(v, new ArrayList<>());
        adjList.get(u).add(v);
        adjList.get(v).add(u);
    }

    // BFS Traversal
    public void bfs(int start) {
        Set<Integer> visited = new HashSet<>();
        Queue<Integer> queue = new LinkedList<>();

        visited.add(start);
        queue.add(start);

        System.out.println("BFS Traversal starting from " + start + ":");
```

```
    while (!queue.isEmpty()) {
        int node = queue.poll();
        System.out.print(node + " ");

        for (int neighbor : adjList.get(node)) {
            if (!visited.contains(neighbor)) {
                visited.add(neighbor);
                queue.add(neighbor);
            }
        }
    }
    System.out.println();
}

// DFS Traversal (using recursion)
public void dfs(int start) {
    Set<Integer> visited = new HashSet<>();
    System.out.println("DFS Traversal starting from " + start + ":");
    dfsRecursive(start, visited);
    System.out.println();
}

// Helper method for DFS (recursive)
private void dfsRecursive(int node, Set<Integer> visited) {
    visited.add(node);
    System.out.print(node + " ");

    for (int neighbor : adjList.get(node)) {
        if (!visited.contains(neighbor)) {
```

```
            dfsRecursive(neighbor, visited);
        }
    }
}
```

Output:-

```
BFS Traversal starting from 0:
0 1 2 3 4 5 6
DFS Traversal starting from 0:
0 1 3 4 2 5 6
```

Find the minimum spanning tree (using any method Kruskal's Algorithm or Prim's Algorithm)

```
import java.util.*;


class Kruskal {
   static class Edge {
      int src, dest, weight;
      Edge(int src, int dest, int weight) {
         this.src = src;
         this.dest = dest;
         this.weight = weight;
      }
   }

   static class DisjointSet {
      int[] parent, rank;

      DisjointSet(int n) {
         parent = new int[n];
         rank = new int[n];
         for (int i = 0; i < n; i++) {
            parent[i] = i;
            rank[i] = 0;
         }
      }

      int find(int i) {
         if (parent[i] != i) {
```

```
            parent[i] = find(parent[i]);
        }
        return parent[i];
    }

    void union(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);
        if (rootX != rootY) {
            if (rank[rootX] < rank[rootY]) {
                parent[rootX] = rootY;
            } else if (rank[rootX] > rank[rootY]) {
                parent[rootY] = rootX;
            } else {
                parent[rootY] = rootX;
                rank[rootX]++;
            }
        }
    }
}

public static List<Edge> kruskalMST(int vertices, List<Edge> edges) {
    List<Edge> result = new ArrayList<>();
    Collections.sort(edges, Comparator.comparingInt(e -> e.weight));
    DisjointSet ds = new DisjointSet(vertices);

    for (Edge edge : edges) {
        int rootSrc = ds.find(edge.src);
        int rootDest = ds.find(edge.dest);
```

```
            if (rootSrc != rootDest) {
                result.add(edge);
                ds.union(rootSrc, rootDest);
            }
        }

        return result;
    }

    public static void main(String[] args) {
        int vertices = 4;
        List<Edge> edges = new ArrayList<>();
        edges.add(new Edge(0, 1, 10));
        edges.add(new Edge(0, 2, 6));
        edges.add(new Edge(0, 3, 5));
        edges.add(new Edge(1, 3, 15));
        edges.add(new Edge(2, 3, 4));

        List<Edge> mst = kruskalMST(vertices, edges);

        System.out.println("Edges in the Minimum Spanning Tree (Kruskal's):");
        for (Edge edge : mst) {
            System.out.println(edge.src + " - " + edge.dest + " : " + edge.weight);
        }
    }
}
```

Output:-

```
PS C:\Users\Yash\OneDrive\Desktop\Java>  & 'C:\Progra
odeDetailsInExceptionMessages' '-cp' 'C:\Users\Yash\A
Edges in the Minimum Spanning Tree (Kruskal's):
2 - 3 : 4
0 - 3 : 5
0 - 1 : 10
PS C:\Users\Yash\OneDrive\Desktop\Java>
```