K.R. MANGALAM UNIVERSITY

THE COMPLETE WORLD OF EDUCATION

NAME : ARYAN

COURSE : B.TECH CSE (FSD)

ROLL NO : 2401360001

CourseName : DataStructure

Faculty : Dr.VandnaBatra

LAB FILE

INDEX

Question 1. Given an array of integers, perform the following operations:

- Traversing
- Insertion
- Deletion

Answer :

Code:

```
public class LAB_MANUAL {

    // Traverse the array
    public static void traverse(int[] arr, int n) {
        System.out.print("Array elements: ");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    // Insert element at a given position
    public static int insertAtPos(int[] arr, int n, int pos, int value)
{
        for (int i = n - 1; i >= pos; i--) {
            arr[i + 1] = arr[i];
        }
        arr[pos] = value;
        n++;
        return n;
    }

    // Delete element at a given position
    public static int deleteAtPos(int[] arr, int n, int pos) {
        for (int i = pos; i < n - 1; i++) {
            arr[i] = arr[i + 1];
        }
        n--;
        return n;
    }
```

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

        int[] arr = new int[100];
        arr[0] = 10;
        arr[1] = 20;
        arr[2] = 30;
        arr[3] = 40;
        arr[4] = 50;
        int n = 5;

        // Traverse
        traverse(arr, n);

        // Insert 99 at position 2
        n = insertAtPos(arr, n, 2, 99);
        traverse(arr, n);

        // Delete element at position 3
        n = deleteAtPos(arr, n, 3);
        traverse(arr, n);
    }
}
```

Output :

```
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>
owCodeDetailsInExceptionMessages' '-cp' 'C:\User
2\redhat.java\jdt_ws\LAB_MANUAL_67c11939\bin' '
Array elements: 10 20 30 40 50
Array elements: 10 20 99 30 40 50
Array elements: 10 20 99 40 50
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>
```

Question 2 : Write a program to implement a singly linked list with methods to insert an element at the head, insert an element at the tail, delete an element by value, and traverse the list to print all elements.

Answer :

Code :

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

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

class SinglyLinkedList {

    Node head;

    // Insert at head
    public void insertAtHead(int value) {
        Node newNode = new Node(value);
        newNode.next = head;
        head = newNode;
    }

    // Insert at tail
    public void insertAtTail(int value) {
        Node newNode = new Node(value);
        if (head == null) {
            head = newNode;
            return;
        }
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }

    // Delete by value
    public void deleteByValue(int value) {
        if (head == null) {
            System.out.println("List is empty.");
            return;
        }
```

```
        if (head.data == value) {
            head = head.next;
            return;
        }

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

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

    // Traverse list
    public void traverse() {
        Node temp = head;
        System.out.print("Linked List: ");
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("NULL");
    }
}

public class QUESTION2 {

    public static void main(String[] args) {

        SinglyLinkedList list = new SinglyLinkedList();

        list.insertAtHead(30);
        list.insertAtHead(20);
        list.insertAtHead(10);

        list.insertAtTail(40);
        list.insertAtTail(50);

        list.traverse();
```

```
        System.out.println("Deleting value 30...");
        list.deleteByValue(30);

        list.traverse();
    }
}
```

Output :

```
[Running] cd "c:\Users\LENOVO\OneDrive\Desktop\LA
Linked List: 10 -> 20 -> 30 -> 40 -> 50 -> NULL
Deleting value 30...
Linked List: 10 -> 20 -> 40 -> 50 -> NULL

[Done] exited with code=0 in 0.798 seconds
```

Question 3 : Write a class to implement a circular linked list with methods to insert an element at the head, insert an element at the tail, delete an element by value, and traverse the list to print all elements.

Answer :

Code :

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

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

class CircularLinkedList {

    Node head;

    // Insert at head
    public void insertAtHead(int value) {
        Node newNode = new Node(value);

        if (head == null) {
            head = newNode;
            newNode.next = head;
            return;
        }

        Node temp = head;
        while (temp.next != head) {
            temp = temp.next;
        }

        newNode.next = head;
        temp.next = newNode;
        head = newNode;
    }

    // Insert at tail
    public void insertAtTail(int value) {
        Node newNode = new Node(value);
```

```
        if (head == null) {
            head = newNode;
            newNode.next = newNode;
            return;
        }

        Node temp = head;
        while (temp.next != head) {
            temp = temp.next;
        }

        temp.next = newNode;
        newNode.next = head;
    }

    // Delete by value
    public void deleteByValue(int value) {

        if (head == null) {
            System.out.println("List is empty.");
            return;
        }

        // Case: deleting head
        if (head.data == value) {
            Node temp = head;

            // if only one node
            if (head.next == head) {
                head = null;
                return;
            }

            // find last node
            while (temp.next != head) {
                temp = temp.next;
            }

            temp.next = head.next;
            head = head.next;
            return;
        }
```

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

        if (temp.next.data == value) {
            temp.next = temp.next.next;
        } else {
            System.out.println("Value not found.");
        }
    }

    // Traverse
    public void traverse() {
        if (head == null) {
            System.out.println("List is empty.");
            return;
        }

        Node temp = head;
        System.out.print("Circular List: ");

        do {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        } while (temp != head);

        System.out.println("(back to head)");
    }
}

public class Question3 {

    public static void main(String[] args) {

        CircularLinkedList list = new CircularLinkedList();

        list.insertAtHead(20);
        list.insertAtHead(10);

        list.insertAtTail(30);
        list.insertAtTail(40);
```

```
        list.traverse();

        System.out.println("Deleting value 30...");
        list.deleteByValue(30);

        list.traverse();
    }
}
```

Output :

```
[Running] cd "c:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL
Circular List: 10 -> 20 -> 30 -> 40 -> (back to head)
Deleting value 30...
Circular List: 10 -> 20 -> 40 -> (back to head)

[Done] exited with code=0 in 0.566 seconds
```

Question 4: Implement a doubly linked list with methods to insert an element at the head, insert an element at the tail, delete an element by value, and reverse the list. Ensure that all operations handle edge cases appropriately.

Answer :

Code :

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

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

class DoublyLinkedList {

    Node head;

    // Insert at head
    public void insertAtHead(int value) {
        Node newNode = new Node(value);

        if (head == null) {
            head = newNode;
            return;
        }

        newNode.next = head;
        head.prev = newNode;
        head = newNode;
    }

    // Insert at tail
    public void insertAtTail(int value) {
        Node newNode = new Node(value);

        if (head == null) {
            head = newNode;
```

```
            return;
        }

        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }

        temp.next = newNode;
        newNode.prev = temp;
    }

    // Delete by value
    public void deleteByValue(int value) {
        if (head == null) {
            System.out.println("List is empty.");
            return;
        }

        // deleting head
        if (head.data == value) {
            if (head.next != null)
                head.next.prev = null;
            head = head.next;
            return;
        }

        Node temp = head;

        while (temp != null && temp.data != value) {
            temp = temp.next;
        }

        if (temp == null) {
            System.out.println("Value not found.");
            return;
        }

        // delete last
        if (temp.next == null) {
            temp.prev.next = null;
        } else {
            temp.prev.next = temp.next;
```

```
            temp.next.prev = temp.prev;
        }
    }

    // Reverse the list
    public void reverse() {
        Node temp = null;
        Node current = head;

        while (current != null) {
            temp = current.prev;
            current.prev = current.next;
            current.next = temp;

            current = current.prev;
        }

        if (temp != null) {
            head = temp.prev;
        }
    }

    // Traverse forward
    public void traverse() {
        Node temp = head;
        System.out.print("Doubly List: ");
        while (temp != null) {
            System.out.print(temp.data + " <-> ");
            temp = temp.next;
        }
        System.out.println("NULL");
    }
}

public class QUESTION4 {

    public static void main(String[] args) {

        DoublyLinkedList list = new DoublyLinkedList();

        list.insertAtHead(20);
        list.insertAtHead(10);
        list.insertAtTail(30);
```

```
        list.insertAtTail(40);

        list.traverse();

        System.out.println("Deleting value 20...");
        list.deleteByValue(20);
        list.traverse();

        System.out.println("Reversing list...");
        list.reverse();
        list.traverse();
    }
}
```

Output :

```
[Running] cd "c:\Users\LENOVO\OneDrive\Desktop\LAB_
Doubly List: 10 <-> 20 <-> 30 <-> 40 <-> NULL
Deleting value 20...
Doubly List: 10 <-> 30 <-> 40 <-> NULL
Reversing list...
Doubly List: 40 <-> 30 <-> 10 <-> NULL

[Done] exited with code=0 in 0.586 seconds
```

Question 5 : Implement a stack using arrays with methods for push, pop, and peek operations.

Code :

```
class StackArray {

    int top;
    int size;
    int[] arr;

    // Constructor
    StackArray(int size) {
        this.size = size;
        arr = new int[size];
        top = -1;
    }

    // Push operation
    public void push(int value) {
        if (top == size - 1) {
            System.out.println("Stack Overflow!");
            return;
        }
        arr[++top] = value;
        System.out.println(value + " pushed into stack");
    }

    // Pop operation
    public int pop() {
        if (top == -1) {
            System.out.println("Stack Underflow!");
            return -1;
        }
        return arr[top--];
    }

    // Peek operation
    public int peek() {
        if (top == -1) {
```

```
            System.out.println("Stack is empty!");
            return -1;
        }
        return arr[top];
    }

    // Display stack
    public void display() {
        if (top == -1) {
            System.out.println("Stack is empty!");
            return;
        }

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

public class QUESTION5 {

    public static void main(String[] args) {

        StackArray stack = new StackArray(5);

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

      System.out.println("Peek: " + stack.peek());

      System.out.println("Popped: " + stack.pop());
      stack.display();

      stack.push(40);
      stack.push(50);
      stack.push(60);  // overflow test
   }
}
```

Output :

```
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.17.10-hotspot\bin\java.exe' '-XX:+Sh
owCodeDetailsInExceptionMessages' '-cp' 'C:\Users\LENOVO\AppData\Roaming\Code\User\workspaceStorage\7a4cfb680ec45e1e98bf729d9447914
2\redhat.java\jdt_ws\LAB_MANUAL_67c11939\bin' 'QUESTION5'
10 pushed into stack
20 pushed into stack
30 pushed into stack
Stack elements: 10 20 30
Peek: 30
Popped: 30
Stack elements: 10 20
40 pushed into stack
50 pushed into stack
60 pushed into stack
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>
```

Question 6 : Write a function to convert an infix expression to a postfix expression using a stack. The function should handle parentheses and operator precedence correctly.

Answer :

```java
import java.util.Stack;

public class QUESTION6 {

    // Function to return precedence of operators
    public static int precedence(char ch) {
        switch (ch) {
            case '+':
            case '-':
                return 1;

            case '*':
            case '/':
                return 2;

            case '^':
                return 3;

            default:
                return -1;
        }
    }

    // Check if operator is right associative (like ^)
    public static boolean isRightAssociative(char ch) {
        return ch == '^';
    }

    // Function to convert infix to postfix
    public static String infixToPostfix(String infix) {
```

```
        Stack<Character> stack = new Stack<>();
        StringBuilder output = new StringBuilder();

        for (char ch : infix.toCharArray()) {

            // If character is operand
            if (Character.isLetterOrDigit(ch)) {
                output.append(ch);
            }
            // If '(' push to stack
            else if (ch == '(') {
                stack.push(ch);
            }
            // If ')' pop until '('
            else if (ch == ')') {
                while (!stack.isEmpty() && stack.peek() != '(') {
                    output.append(stack.pop());
                }
                stack.pop(); // remove '('
            }
            // Operator encountered
            else {
                while (!stack.isEmpty() &&
                        precedence(ch) <= precedence(stack.peek()) &&
                        !isRightAssociative(ch)) {

                    output.append(stack.pop());
                }
                stack.push(ch);
            }
        }

        // Pop remaining operators
        while (!stack.isEmpty()) {
            output.append(stack.pop());
        }

        return output.toString();
    }

    public static void main(String[] args) {

        String infix = "A*(B+C)/D";
```

```
        System.out.println("Infix Expression: " + infix);
        System.out.println("Postfix Expression: " +
infixToPostfix(infix));
    }
}
```

Output :

```
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.17.10-hotspot\bin\java.exe' '-XX:+Sh
owCodeDetailsInExceptionMessages' '-cp' 'C:\Users\LENOVO\AppData\Roaming\Code\User\workspaceStorage\7a4cfb680ec45e1e98bf729d9447914
2\redhat.java\jdt_ws\LAB_MANUAL_67c11939\bin' 'QUESTION6'
Infix Expression: A*(B+C)/D
Postfix Expression: ABC+*D/
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>
```

Question 7 :Create a linear queue using an array with methods for enqueue, dequeue, and checking if the queue is empty or full.

Answer :

```
class LinearQueue {

    int front, rear, size;
    int[] arr;

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

    // Enqueue operation
    public void enqueue(int value) {
        if (rear == size - 1) {
            System.out.println("Queue Overflow!");
            return;
        }
        arr[++rear] = value;
        System.out.println(value + " enqueued");
    }

    // Dequeue operation
    public int dequeue() {
        if (front > rear) {
            System.out.println("Queue Underflow!");
            return -1;
        }
        return arr[front++];
    }

    // Peek operation
```

```
    public int peek() {
        if (front > rear) {
            System.out.println("Queue is empty!");
            return -1;
        }
        return arr[front];
    }

    // Display operation
    public void display() {
        if (front > rear) {
            System.out.println("Queue is empty!");
            return;
        }

        System.out.print("Queue elements: ");
        for (int i = front; i <= rear; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
}

public class QUESTION7 {

    public static void main(String[] args) {

        LinearQueue queue = new LinearQueue(5);

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

        queue.display();

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

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

        queue.enqueue(40);
        queue.enqueue(50);
        queue.enqueue(60);  // Overflow
```

```
        }
}
```

Output :

```
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.17.10-hotspot\bin\java.exe' '-XX:+Sh
owCodeDetailsInExceptionMessages' '-cp' 'C:\Users\LENOVO\AppData\Roaming\Code\User\workspaceStorage\7a4cfb680ec45e1e98bf729d9447914
2\redhat.java\jdt_ws\LAB_MANUAL_67c11939\bin' 'QUESTION7'
10 enqueued
20 enqueued
30 enqueued
Queue elements: 10 20 30
Peek: 10
Dequeued: 10
Queue elements: 20 30
40 enqueued
50 enqueued
Queue Overflow!
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>
```

QUESTION 8 : Create a circular queue using array with methods for enqueue, dequeue, and checking if the queue is empty or full. Ensure that the circular nature of the queue is maintained after each operation.

Answer :

```
class CircularQueue {

    int front, rear, size;
    int[] arr;

    CircularQueue(int size) {
        this.size = size;
        arr = new int[size];
        front = -1;
        rear = -1;
    }

    // Check if queue is full
    boolean isFull() {
        return (front == 0 && rear == size - 1) || (rear + 1 == front);
    }

    // Check if queue is empty
    boolean isEmpty() {
        return front == -1;
    }

    // Enqueue operation
    public void enqueue(int value) {
        if (isFull()) {
            System.out.println("Queue Overflow!");
            return;
        }

        if (front == -1) { // first element
            front = 0;
        }
```

```
        rear = (rear + 1) % size;
        arr[rear] = value;
        System.out.println(value + " enqueued");
    }

    // Dequeue operation
    public int dequeue() {
        if (isEmpty()) {
            System.out.println("Queue Underflow!");
            return -1;
        }

        int data = arr[front];

        if (front == rear) { // last element
            front = -1;
            rear = -1;
        } else {
            front = (front + 1) % size;
        }

        return data;
    }

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

    // Display queue
    public void display() {
        if (isEmpty()) {
            System.out.println("Queue is empty!");
            return;
        }

        System.out.print("Circular Queue: ");
        int i = front;
```

```
        while (true) {
            System.out.print(arr[i] + " ");
            if (i == rear)
                break;
            i = (i + 1) % size;
        }

        System.out.println();
    }
}

public class QUESTION8 {

    public static void main(String[] args) {

        CircularQueue cq = new CircularQueue(5);

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

        cq.display();

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

        cq.enqueue(40);
        cq.enqueue(50);
        cq.enqueue(60);  // wrap around
        cq.display();

        cq.enqueue(70);  // Overflow
    }
}
```

Output :

```
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.17.10-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\LENOVO\AppData\Roaming\Code\User\workspaceStorage\7a4cfb680ec45e1e98bf729d94479142\redhat.java\jdt_ws\LAB_MANUAL_67c11939\bin' 'QUESTION8'
10 enqueued
20 enqueued
30 enqueued
Circular Queue: 10 20 30
Dequeued: 10
Circular Queue: 20 30
40 enqueued
50 enqueued
60 enqueued
Circular Queue: 20 30 40 50 60
Queue Overflow!
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>
```

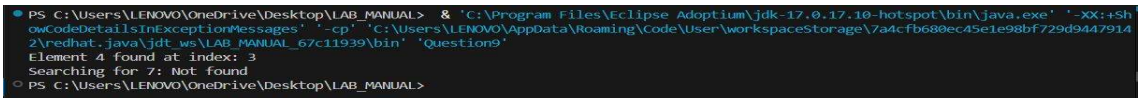
QUESTION 9 : Implement linear search

ANSWER :

```java
public class Question9 {

    // Iterative linear search: returns index or -1
    public static int linearSearchIterative(int[] arr, int key) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == key) {
                return i; // found at index i
            }
        }
        return -1; // not found
    }

    public static void main(String[] args) {
        int[] arr = {5, 3, 8, 4, 2, 9, 1};
        int key = 4;

        int idx = linearSearchIterative(arr, key);
        if (idx != -1) {
            System.out.println("Element " + key + " found at index: " + idx);
        } else {
            System.out.println("Element " + key + " not found in the array.");
        }

        // test not-found case
        key = 7;
        idx = linearSearchIterative(arr, key);
```

```
        System.out.println("Searching for " + key + ": " + (idx == -1 ?
"Not found" : "Found at index " + idx));
    }
}
```

Output :



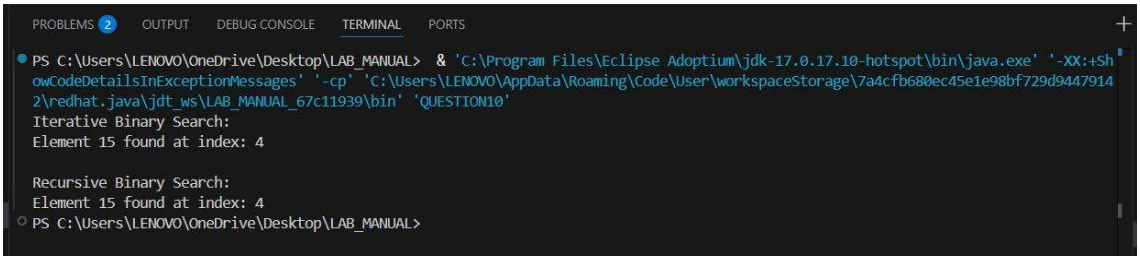
Question 10 : Implement binary search(iterative and recursive)

Answer :

```
public class QUESTION10 {

    // Iterative Binary Search
    public static int binarySearchIterative(int[] arr, int key) {
        int left = 0;
        int right = arr.length - 1;

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

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

        return -1;
    }

    // Recursive Binary Search
    public static int binarySearchRecursive(int[] arr, int left, int
right, int key) {

        if (left > right) {
```

```
            return -1;
        }

        int mid = (left + right) / 2;

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

        if (key < arr[mid]) {
            return binarySearchRecursive(arr, left, mid - 1, key);
        } else {
            return binarySearchRecursive(arr, mid + 1, right, key);
        }
    }

    public static void main(String[] args) {

        int[] arr = {2, 5, 7, 10, 15, 20, 25, 30};
        int key = 15;

        System.out.println("Iterative Binary Search:");
        int idx1 = binarySearchIterative(arr, key);
        System.out.println("Element " + key + " found at index: " +
idx1);

        System.out.println("\nRecursive Binary Search:");
        int idx2 = binarySearchRecursive(arr, 0, arr.length - 1, key);
        System.out.println("Element " + key + " found at index: " +
idx2);
    }
}
```

Output :



Question 11: Implement various sorting algorithms including Insertion sort, selection sort, bubble sort, and analyze their performance on different input sizes.

Answer :

```java
public class QUESTION11 {

    // ---------------------- BUBBLE SORT -------------------------
    public static void bubbleSort(int[] arr) {
        int n = arr.length;

        for (int i = 0; i < n - 1; i++) {
            boolean swapped = false;

            for (int j = 0; j < n - i - 1; j++) {

                if (arr[j] > arr[j + 1]) {
                    // swap
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;

                    swapped = true;
                }
            }

            if (!swapped) break; // Already sorted
        }
```

```java
    }

    // ---------------------- SELECTION SORT -------------------------
    public static void selectionSort(int[] arr) {
        int n = arr.length;

        for (int i = 0; i < n - 1; i++) {

            int minIndex = i;

            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex])
                    minIndex = j;
            }

            // swap
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }

    // ---------------------- INSERTION SORT -------------------------
    public static void insertionSort(int[] arr) {
        int n = arr.length;

        for (int i = 1; i < n; i++) {

            int key = arr[i];
            int j = i - 1;

            // Move elements greater than key ahead
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }

            arr[j + 1] = key;
        }
    }


    // ---------------------- PRINT ARRAY -------------------------
```

```java
    public static void printArray(String msg, int[] arr) {
        System.out.print(msg + ": ");
        for (int x : arr) System.out.print(x + " ");
        System.out.println();
    }


    // ---------------------- MAIN -------------------------
    public static void main(String[] args) {

        int[] arr1 = {5, 2, 9, 1, 5, 6};
        int[] arr2 = arr1.clone();
        int[] arr3 = arr1.clone();

        System.out.println("Original Array:");
        printArray("Array", arr1);

        bubbleSort(arr1);
        printArray("Bubble Sorted", arr1);

        selectionSort(arr2);
        printArray("Selection Sorted", arr2);

        insertionSort(arr3);
        printArray("Insertion Sorted", arr3);
    }
}
```

Output :

```
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.17.10-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\LENOVO\AppData\Roaming\Code\User\workspaceStorage\7a4cfb680ec45e1e98bf729d94479142\redhat.java\jdt_ws\LAB_MANUAL_67c11939\bin' 'QUESTION11'
Original Array:
Array: 5 2 9 1 5 6
Bubble Sorted: 1 2 5 5 6 9
Selection Sorted: 1 2 5 5 6 9
Insertion Sorted: 1 2 5 5 6 9
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>
```

QUESTION 12 : Implement various sorting algorithms including Quick Sort, Merge Sort, Heap Sort, and analyze their performance on different input sizes. Ensure the implementation handles edge cases such as duplicate values and nearly sorted arrays.

Answer :

```
public class QUESTION12 {

    // ---------------------- QUICK SORT -------------------------
    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {

            int pivotIndex = partition(arr, low, high);

            quickSort(arr, low, pivotIndex - 1);
            quickSort(arr, pivotIndex + 1, high);
        }
    }

    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;

        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                // swap
```

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

    // swap pivot
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    return i + 1;
}


// ---------------------- MERGE SORT -------------------------
public static void mergeSort(int[] arr, int left, int right) {
    if (left < right) {

        int mid = (left + right) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

public static void merge(int[] arr, int left, int mid, int right) {

    int n1 = mid - left + 1;
    int n2 = right - mid;

    int[] L = new int[n1];
    int[] R = new int[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];

    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    int i = 0, j = 0;
```

```
        int k = left;

        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k++] = L[i++];
            } else {
                arr[k++] = R[j++];
            }
        }

        while (i < n1)
            arr[k++] = L[i++];

        while (j < n2)
            arr[k++] = R[j++];
    }


    // ---------------------- HEAP SORT -------------------------
    public static void heapSort(int[] arr) {
        int n = arr.length;

        // build max heap
        for (int i = n / 2 - 1; i >= 0; i--) {
            heapify(arr, n, i);
        }

        // extract elements
        for (int i = n - 1; i >= 0; i--) {

            // move max to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            heapify(arr, i, 0);
        }
    }

    public static void heapify(int[] arr, int size, int i) {

        int largest = i;
        int left = 2 * i + 1;
```

```
        int right = 2 * i + 2;

        if (left < size && arr[left] > arr[largest])
            largest = left;

        if (right < size && arr[right] > arr[largest])
            largest = right;

        if (largest != i) {
            int temp = arr[i];
            arr[i] = arr[largest];
            arr[largest] = temp;

            heapify(arr, size, largest);
        }
    }


    // ---------------------- PRINT ARRAY -------------------------
    public static void printArray(String msg, int[] arr) {
        System.out.print(msg + ": ");
        for (int x : arr)
            System.out.print(x + " ");
        System.out.println();
    }


    // ---------------------- MAIN -------------------------
    public static void main(String[] args) {

        int[] arr1 = {9, 3, 7, 1, 5, 4, 9};
        int[] arr2 = arr1.clone();
        int[] arr3 = arr1.clone();

        printArray("Original", arr1);

        quickSort(arr1, 0, arr1.length - 1);
        printArray("Quick Sort", arr1);

        mergeSort(arr2, 0, arr2.length - 1);
        printArray("Merge Sort", arr2);

        heapSort(arr3);
```

```
        printArray("Heap Sort", arr3);
    }
}
```

Output :

```
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.17.10-hotspot\bin\java.exe' '-XX:+Sh
owCodeDetailsInExceptionMessages' '-cp' 'C:\Users\LENOVO\AppData\Roaming\Code\User\workspaceStorage\7a4cfb680ec45e1e98bf729d9447914
2\redhat.java\jdt_ws\LAB_MANUAL_67c11939\bin' 'QUESTION12'
Original: 9 3 7 1 5 4 9
Quick Sort: 1 3 4 5 7 9 9
Merge Sort: 1 3 4 5 7 9 9
Heap Sort: 1 3 4 5 7 9 9
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>
```

QUESTION 13 : Given preorder and inorder traversal of a tree, construct the binary tree.

Answer :

```
import java.util.*;

class Node {
    int data;
    Node left, right;

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

public class QUESTION13 {

    static int preorderIndex = 0;
    static Map<Integer, Integer> inorderMap = new HashMap<>();

    // Build Tree Function
    public static Node buildTree(int[] preorder, int[] inorder, int
start, int end) {

        // No elements left
        if (start > end)
```

```
            return null;

        // Pick current node from preorder
        int rootValue = preorder[preorderIndex++];
        Node root = new Node(rootValue);

        // Leaf node check
        if (start == end)
            return root;

        // Find root position in inorder
        int inorderIndex = inorderMap.get(rootValue);

        // Build left and right subtrees
        root.left = buildTree(preorder, inorder, start, inorderIndex -
1);
        root.right = buildTree(preorder, inorder, inorderIndex + 1,
end);

        return root;
    }

    // Print inorder (to verify)
    public static void printInorder(Node root) {
        if (root != null) {
            printInorder(root.left);
            System.out.print(root.data + " ");
            printInorder(root.right);
        }
    }

    public static void main(String[] args) {

        int[] preorder = {3, 9, 20, 15, 7};
        int[] inorder = {9, 3, 15, 20, 7};

        // Map inorder values for quick lookup
        for (int i = 0; i < inorder.length; i++) {
            inorderMap.put(inorder[i], i);
        }

        Node root = buildTree(preorder, inorder, 0, inorder.length - 1);
```

```
        System.out.println("Tree constructed. Inorder traversal of
tree:");
        printInorder(root);
    }
}
```

Output :

```
Tree constructed. Inorder traversal of tree:
9 3 15 20 7
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>
```

Question 14 : Perform the traversal of graph(DFS,BFS)

Answer :

```
import java.util.*;

public class QUESTION14 {

    // Graph using adjacency list
    static class Graph {
        int vertices;
        LinkedList<Integer>[] adj;

        Graph(int v) {
            vertices = v;
            adj = new LinkedList[v];

            for (int i = 0; i < v; i++) {
                adj[i] = new LinkedList<>();
            }
        }

        // Add edge (undirected graph)
        void addEdge(int src, int dest) {
            adj[src].add(dest);
            adj[dest].add(src);
        }

        // BFS Traversal
        void BFS(int start) {
            boolean[] visited = new boolean[vertices];
            Queue<Integer> queue = new LinkedList<>();

            visited[start] = true;
            queue.add(start);
```

```
        System.out.print("BFS Traversal: ");

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

            for (int neighbor : adj[node]) {
                if (!visited[neighbor]) {
                    visited[neighbor] = true;
                    queue.add(neighbor);
                }
            }
        }
        System.out.println();
    }

    // DFS Traversal
    void DFS(int start) {
        boolean[] visited = new boolean[vertices];
        System.out.print("DFS Traversal: ");
        DFSUtil(start, visited);
        System.out.println();
    }

    void DFSUtil(int node, boolean[] visited) {
        visited[node] = true;
        System.out.print(node + " ");

        for (int neighbor : adj[node]) {
            if (!visited[neighbor]) {
                DFSUtil(neighbor, visited);
            }
        }
    }
}

public static void main(String[] args) {

    Graph g = new Graph(6);

    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 5);

    g.BFS(0);
    g.DFS(0);
```

```
    }
}
```

Output :

```
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.17.10-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\LENOVO\AppData\Roaming\Code\User\workspaceStorage\7a4cfb680ec45e1e98bf729d94479142\redhat.java\jdt_ws\LAB_MANUAL_67c11939\bin' 'QUESTION14'
BFS Traversal: 0 1 2 3 4 5
DFS Traversal: 0 1 3 4 2 5
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>
```

Question 15 : Implement Prim's and Kruskal's algorithm to find the minimum spanning tree of a graph.

Answer :

```
import java.util.*;

public class QUESTION15 {

    // ------------------------------------------------------------
    //                 PRIM'S ALGORITHM  (Adjacency Matrix)
    // ------------------------------------------------------------
    static int V = 5;

    int minKey(int key[], boolean mstSet[]) {
        int min = Integer.MAX_VALUE, minIndex = -1;

        for (int v = 0; v < V; v++)
            if (!mstSet[v] && key[v] < min) {
                min = key[v];
                minIndex = v;
            }

        return minIndex;
    }

    void primMST(int graph[][]) {
        int parent[] = new int[V];
        int key[] = new int[V];
        boolean mstSet[] = new boolean[V];

        for (int i = 0; i < V; i++) {
            key[i] = Integer.MAX_VALUE;
```

```
            mstSet[i] = false;
        }

        key[0] = 0;
        parent[0] = -1;

        for (int count = 0; count < V - 1; count++) {
            int u = minKey(key, mstSet);
            mstSet[u] = true;

            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];
                }
        }

        System.out.println("Prim's MST:");
        for (int i = 1; i < V; i++)
            System.out.println(parent[i] + " - " + i + " : " +
graph[i][parent[i]]);
    }


    // ------------------------------------------------------------
    //                KRUSKAL'S ALGORITHM  (Edge List)
    // ------------------------------------------------------------
    static class Edge implements Comparable<Edge> {
        int src, dest, weight;

        public int compareTo(Edge other) {
            return this.weight - other.weight;
        }
    }

    static class Subset {
        int parent, rank;
    }

    int find(Subset subsets[], int i) {
        if (subsets[i].parent != i)
            subsets[i].parent = find(subsets, subsets[i].parent);
```

```
        return subsets[i].parent;
    }

    void union(Subset subsets[], int x, int y) {
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);

        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++;
        }
    }

    void kruskalMST(Edge edges[], int V, int E) {

        Arrays.sort(edges);

        Edge result[] = new Edge[V];
        int e = 0;
        int i = 0;

        Subset subsets[] = new Subset[V];
        for (int v = 0; v < V; v++) {
            subsets[v] = new Subset();
            subsets[v].parent = v;
            subsets[v].rank = 0;
        }

        while (e < V - 1) {
            Edge next = edges[i++];
            int x = find(subsets, next.src);
            int y = find(subsets, next.dest);

            if (x != y) {
                result[e++] = next;
                union(subsets, x, y);
            }
        }
```

```
        System.out.println("\nKruskal's MST:");
        for (i = 0; i < e; i++)
            System.out.println(result[i].src + " - " + result[i].dest +
" : " + result[i].weight);
    }


    // ------------------------------------------------------------
    //                       MAIN FUNCTION
    // ------------------------------------------------------------
    public static void main(String[] args) {

        QUESTION15 obj = new QUESTION15();

        int graph[][] = {
                {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}
        };

        obj.primMST(graph);

        Edge edges[] = {
                new Edge(), new Edge(), new Edge(), new Edge(), new
Edge(),
                new Edge(), new Edge(), new Edge(), new Edge()
        };

        edges[0].src = 0; edges[0].dest = 1; edges[0].weight = 2;
        edges[1].src = 1; edges[1].dest = 2; edges[1].weight = 3;
        edges[2].src = 0; edges[2].dest = 3; edges[2].weight = 6;
        edges[3].src = 1; edges[3].dest = 3; edges[3].weight = 8;
        edges[4].src = 1; edges[4].dest = 4; edges[4].weight = 5;
        edges[5].src = 2; edges[5].dest = 4; edges[5].weight = 7;
        edges[6].src = 3; edges[6].dest = 4; edges[6].weight = 9;

        obj.kruskalMST(edges, 5, 7);
    }
}
```

Output :

```
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.17.10-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\LENOVO\AppData\Roaming\Code\User\workspaceStorage\7a4cfb680ec45e1e98bf729d94479142\redhat.java\jdt_ws\LAB_MANUAL_67c11939\bin' 'QUESTION15'
Prim's MST:
0 - 1 : 2
1 - 2 : 3
0 - 3 : 6
1 - 4 : 5

Kruskal's MST:
0 - 1 : 2
1 - 2 : 3
1 - 4 : 5
0 - 3 : 6
```

QUESTION 16 : Implement Dijkstra's algorithm to find the shortest path from a source vertex to all other vertices in a weighted graph. Use both adjacency matrix and adjacency list representations for the graph. Ensure the algorithm handles negative weights appropriately.

Answer :

```java
import java.util.*;

public class QUESTION16 {

    // ------------------------------------------------------------
    //   CHECK FOR NEGATIVE WEIGHTS (COMMON FOR BOTH)
    // ------------------------------------------------------------
    static boolean containsNegative(int[][] graph) {
        for (int[] row : graph) {
            for (int w : row) {
                if (w < 0) return true;
            }
        }
        return false;
    }

    // ------------------------------------------------------------
    //              DIJKSTRA USING ADJACENCY MATRIX
    // ------------------------------------------------------------
    static void dijkstraMatrix(int[][] graph, int src) {

        int V = graph.length;

        if (containsNegative(graph)) {
            System.out.println("Error: Graph contains negative weights.
Dijkstra not possible.");
```

```
            return;
        }

        int[] dist = new int[V];
        boolean[] visited = new boolean[V];

        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[src] = 0;

        for (int count = 0; count < V - 1; count++) {

            int u = -1;
            int minDist = Integer.MAX_VALUE;

            for (int i = 0; i < V; i++) {
                if (!visited[i] && dist[i] < minDist) {
                    minDist = dist[i];
                    u = i;
                }
            }

            visited[u] = true;

            for (int v = 0; v < V; v++) {

                if (!visited[v] && graph[u][v] != 0 &&
                        dist[u] != Integer.MAX_VALUE &&
                        dist[u] + graph[u][v] < dist[v]) {

                    dist[v] = dist[u] + graph[u][v];
                }
            }
        }

        System.out.println("\nDijkstra (Adjacency Matrix):");
        for (int i = 0; i < V; i++)
            System.out.println("Distance from " + src + " to " + i + "
= " + dist[i]);
    }


    // ------------------------------------------------------------
    //          DIJKSTRA USING ADJACENCY LIST (PQ)
```

```java
    // ------------------------------------------------------------
    static class Pair {
        int node, dist;
        Pair(int n, int d) { node = n; dist = d; }
    }

    static void dijkstraList(List<List<Pair>> graph, int src) {

        int V = graph.size();

        int[] dist = new int[V];
        Arrays.fill(dist, Integer.MAX_VALUE);

        dist[src] = 0;

        PriorityQueue<Pair> pq = new
PriorityQueue<>(Comparator.comparingInt(a -> a.dist));
        pq.add(new Pair(src, 0));

        System.out.println("\nDijkstra (Adjacency List):");

        while (!pq.isEmpty()) {
            Pair p = pq.poll();
            int u = p.node;

            for (Pair edge : graph.get(u)) {

                if (edge.dist < 0) {
                    System.out.println("Error: Graph contains negative
weights. Dijkstra not possible.");
                    return;
                }

                if (dist[u] + edge.dist < dist[edge.node]) {
                    dist[edge.node] = dist[u] + edge.dist;
                    pq.add(new Pair(edge.node, dist[edge.node]));
                }
            }
        }

        for (int i = 0; i < V; i++)
            System.out.println("Distance from " + src + " to " + i + "
= " + dist[i]);
```

```java
    }


    // ------------------------------------------------------------
    //                          MAIN
    // ------------------------------------------------------------
    public static void main(String[] args) {

        // ----------------- MATRIX GRAPH -----------------
        int[][] matrixGraph = {
                {0, 10, 0, 30, 100},
                {10, 0, 50, 0, 0},
                {0, 50, 0, 20, 10},
                {30, 0, 20, 0, 60},
                {100, 0, 10, 60, 0}
        };

        dijkstraMatrix(matrixGraph, 0);


        // ------------------ LIST GRAPH ------------------
        int V = 5;
        List<List<Pair>> graph = new ArrayList<>();

        for (int i = 0; i < V; i++)
            graph.add(new ArrayList<>());

        // Add edges (undirected)
        graph.get(0).add(new Pair(1, 10));
        graph.get(0).add(new Pair(3, 30));
        graph.get(0).add(new Pair(4, 100));

        graph.get(1).add(new Pair(0, 10));
        graph.get(1).add(new Pair(2, 50));

        graph.get(2).add(new Pair(1, 50));
        graph.get(2).add(new Pair(3, 20));
        graph.get(2).add(new Pair(4, 10));

        graph.get(3).add(new Pair(0, 30));
        graph.get(3).add(new Pair(2, 20));
        graph.get(3).add(new Pair(4, 60));
```

```
        graph.get(4).add(new Pair(0, 100));
        graph.get(4).add(new Pair(2, 10));
        graph.get(4).add(new Pair(3, 60));

        dijkstraList(graph, 0);
    }
}
```

Output :

```
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.17.10-hotspot\bin\java.exe' '-XX:+Sh
owCodeDetailsInExceptionMessages' '-cp' 'C:\Users\LENOVO\AppData\Roaming\Code\User\workspaceStorage\7a4cfb680ec45e1e98bf729d9447914
2\redhat.java\jdt_ws\LAB_MANUAL_67c11939\bin' 'QUESTION16'

Dijkstra (Adjacency Matrix):
Distance from 0 to 0 = 0
Distance from 0 to 1 = 10
Distance from 0 to 2 = 50
Distance from 0 to 3 = 30
Distance from 0 to 4 = 60

Dijkstra (Adjacency List):
Distance from 0 to 0 = 0
Distance from 0 to 1 = 10
Distance from 0 to 2 = 50
Distance from 0 to 3 = 30
Distance from 0 to 4 = 60
PS C:\Users\LENOVO\OneDrive\Desktop\LAB_MANUAL>
```