K.R. MANGALAM UNIVERSITY

THE COMPLETE WORLD OF EDUCATION

NAME – SUBHRAJEET DASH

COURSE – BTECH CSE (FULL STACK)

YEAR – 2024 - 2028

SUBJECT – DATA STUCTURE

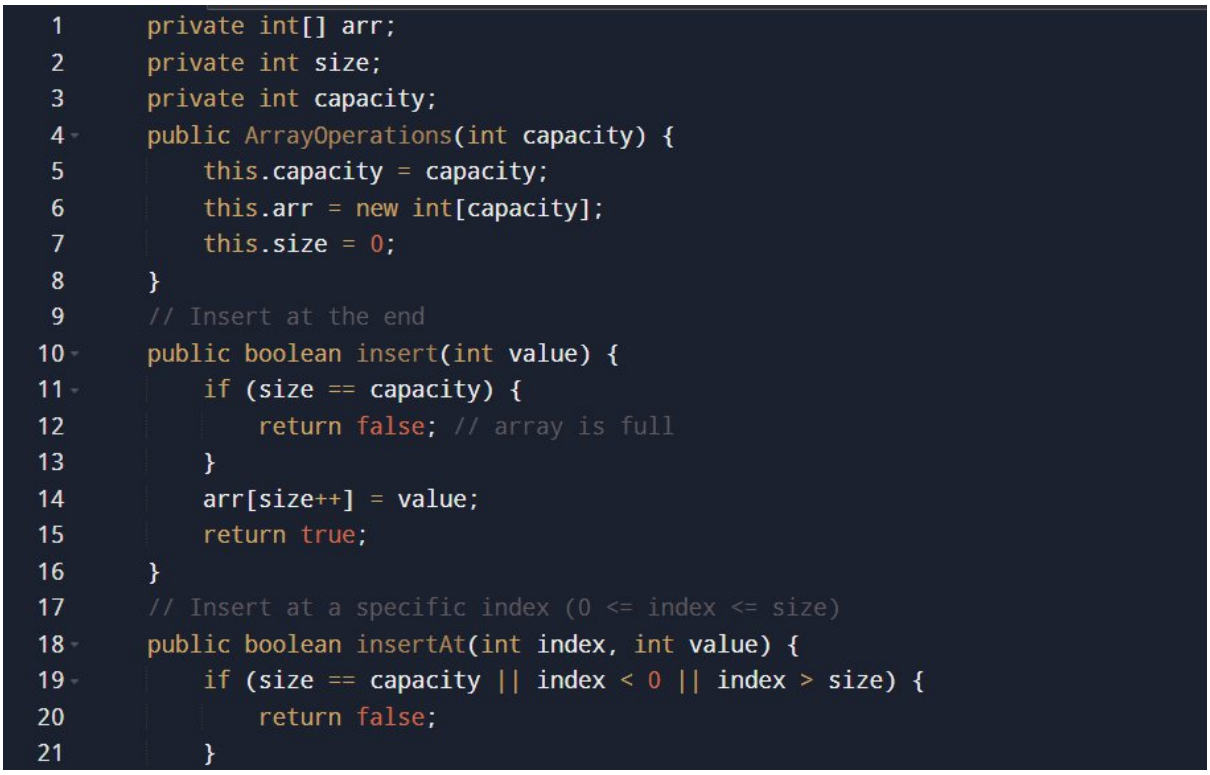
ROLL – 2401350018

LAB MANUAL FILE

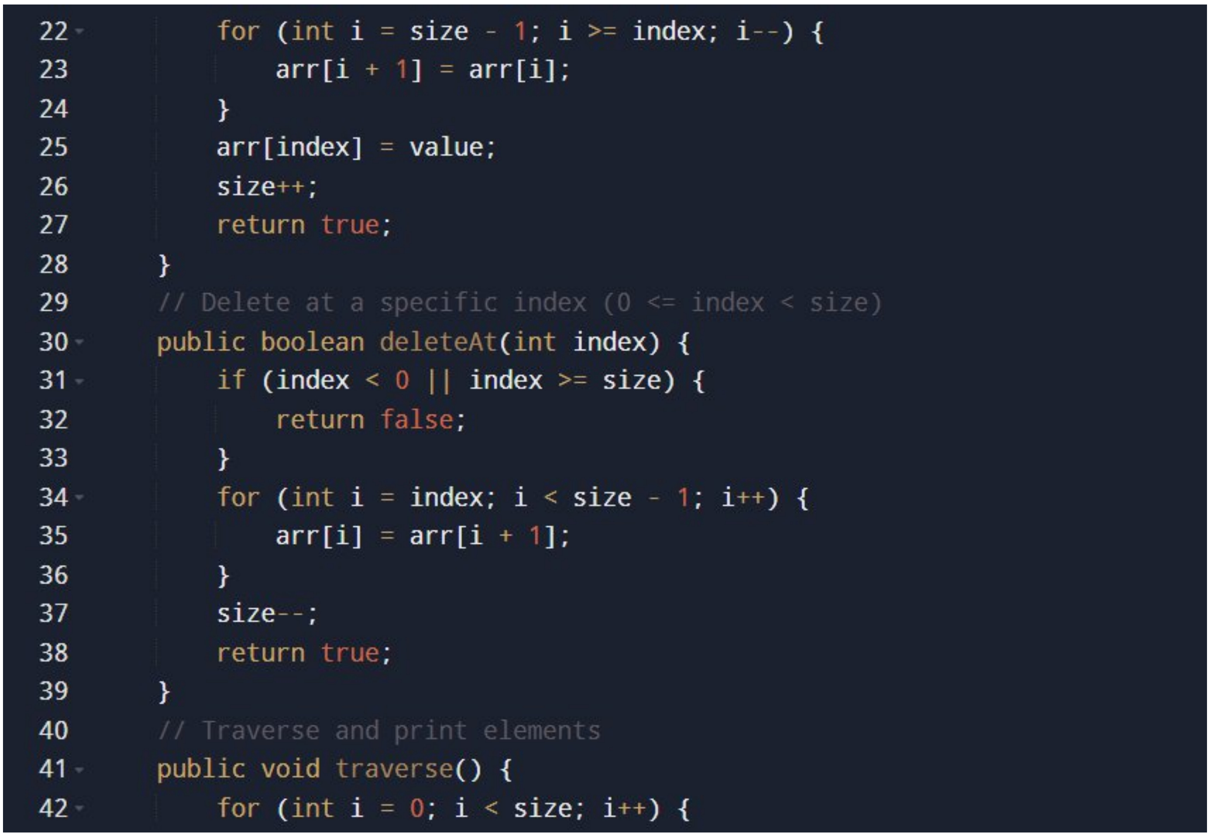
SUBMITTED TO : VANDANA MAM

Q1. Given an array of integers, perform the following operations: traversing , insertion, deletion

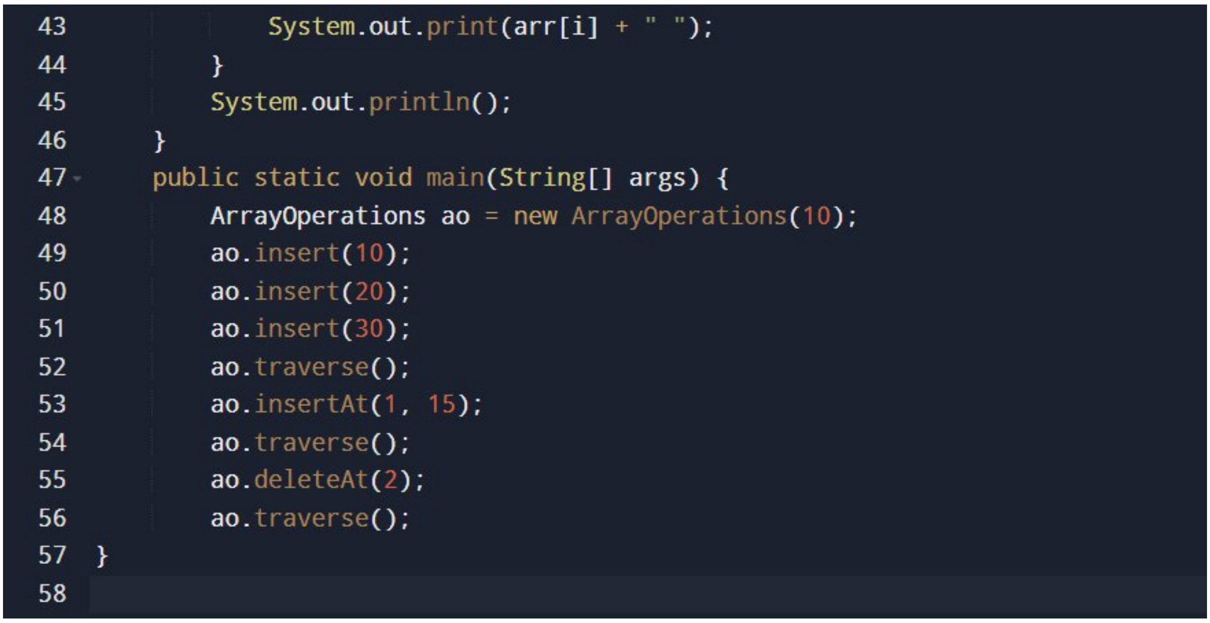
CODE :



```java
private int[] arr;
private int size;
private int capacity;
public ArrayOperations(int capacity) {
    this.capacity = capacity;
    this.arr = new int[capacity];
    this.size = 0;
}
// Insert at the end
public boolean insert(int value) {
    if (size == capacity) {
        return false; // array is full
    }
    arr[size++] = value;
    return true;
}
// Insert at a specific index (0 <= index <= size)
public boolean insertAt(int index, int value) {
    if (size == capacity || index < 0 || index > size) {
        return false;
    }
```

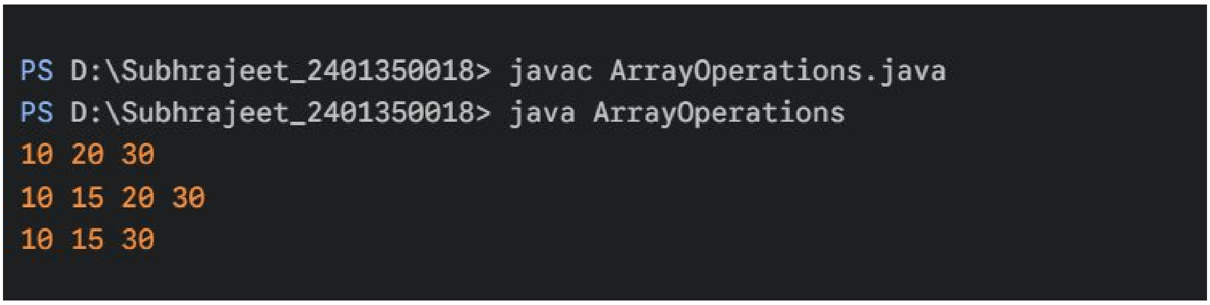


```java
    for (int i = size - 1; i >= index; i--) {
        arr[i + 1] = arr[i];
    }
    arr[index] = value;
    size++;
    return true;
}
// Delete at a specific index (0 <= index < size)
public boolean deleteAt(int index) {
    if (index < 0 || index >= size) {
        return false;
    }
    for (int i = index; i < size - 1; i++) {
        arr[i] = arr[i + 1];
    }
    size--;
    return true;
}
// Traverse and print elements
public void traverse() {
    for (int i = 0; i < size; i++) {
```



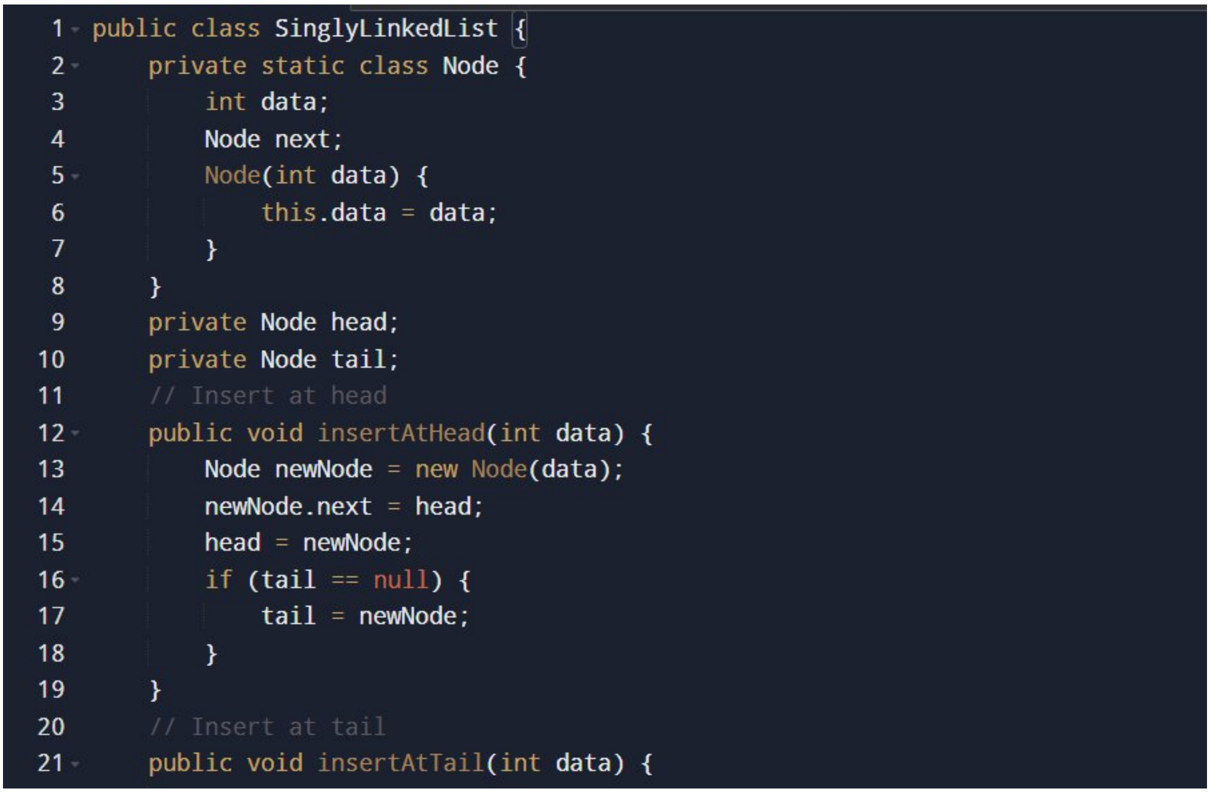
```java
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        ArrayOperations ao = new ArrayOperations(10);
        ao.insert(10);
        ao.insert(20);
        ao.insert(30);
        ao.traverse();
        ao.insertAt(1, 15);
        ao.traverse();
        ao.deleteAt(2);
        ao.traverse();
}

```

OUTPUT :

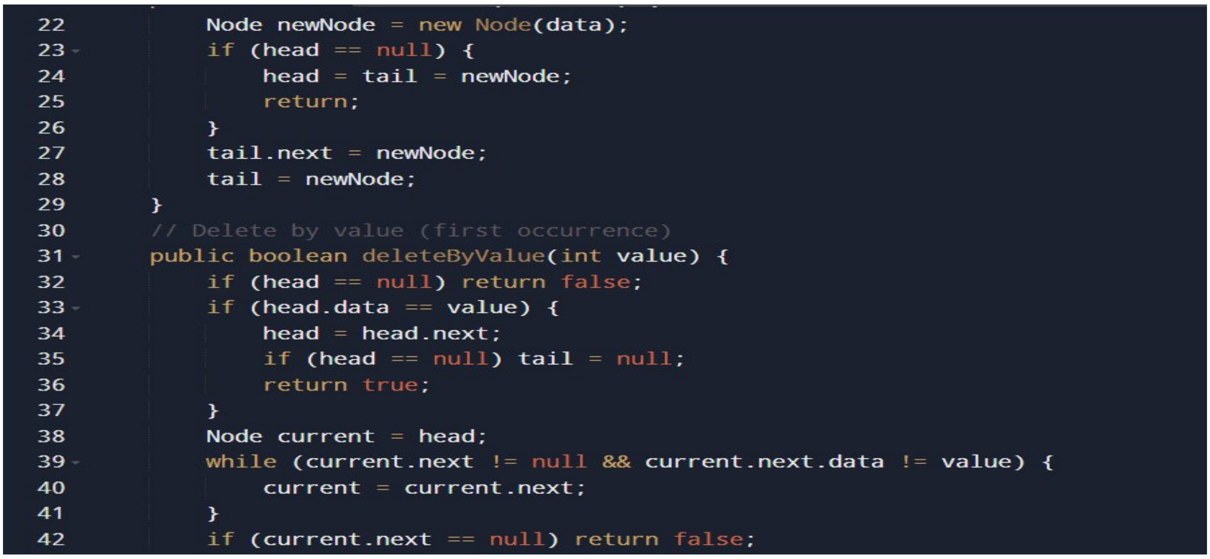


```
PS D:\Subhrajeet_2401350018> javac ArrayOperations.java
PS D:\Subhrajeet_2401350018> java ArrayOperations
10 20 30
10 15 20 30
10 15 30
```

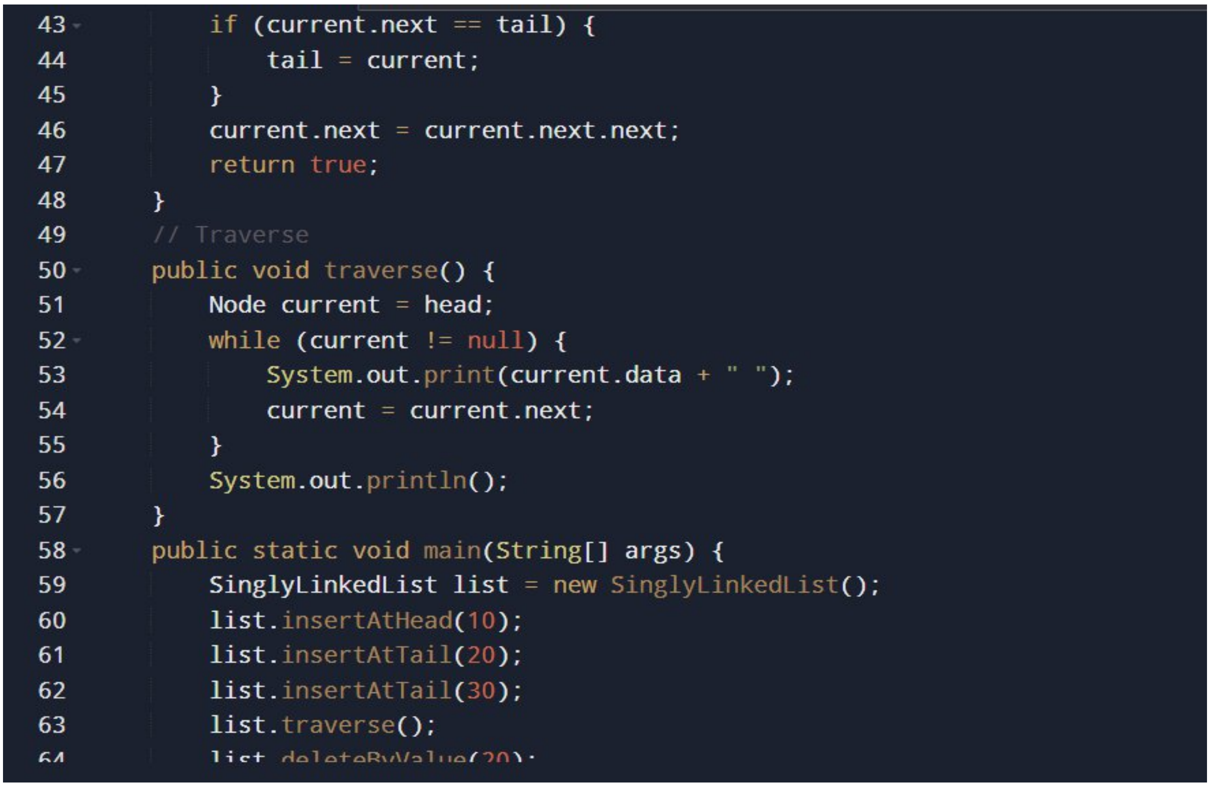
Q2. Write a program to implement a singly linked list with methods to insert an element at the head, insert an element at the tail, delete an element by value, and traverse the list to print all elements.



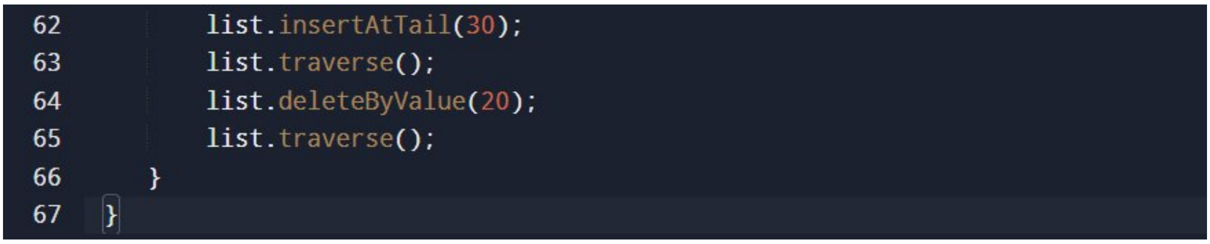
```java
public class SinglyLinkedList {
    private static class Node {
        int data;
        Node next;
        Node(int data) {
            this.data = data;
        }
    }
    private Node head;
    private Node tail;
    // Insert at head
    public void insertAtHead(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
        if (tail == null) {
            tail = newNode;
        }
    }
    // Insert at tail
    public void insertAtTail(int data) {
```



```java
        Node newNode = new Node(data);
        if (head == null) {
            head = tail = newNode;
            return;
        }
        tail.next = newNode;
        tail = newNode;
    }
    // Delete by value (first occurrence)
    public boolean deleteByValue(int value) {
        if (head == null) return false;
        if (head.data == value) {
            head = head.next;
            if (head == null) tail = null;
            return true;
        }
        Node current = head;
        while (current.next != null && current.next.data != value) {
            current = current.next;
        }
        if (current.next == null) return false;
```

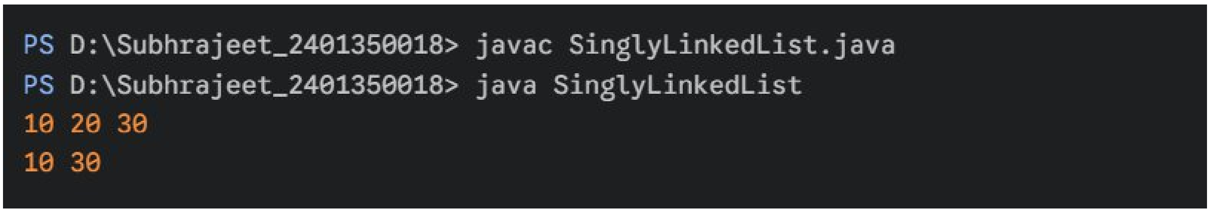


```
        if (current.next == tail) {
            tail = current;
        }
        current.next = current.next.next;
        return true;
    }
    // Traverse
    public void traverse() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        SinglyLinkedList list = new SinglyLinkedList();
        list.insertAtHead(10);
        list.insertAtTail(20);
        list.insertAtTail(30);
        list.traverse();
        list.deleteByValue(20);
```



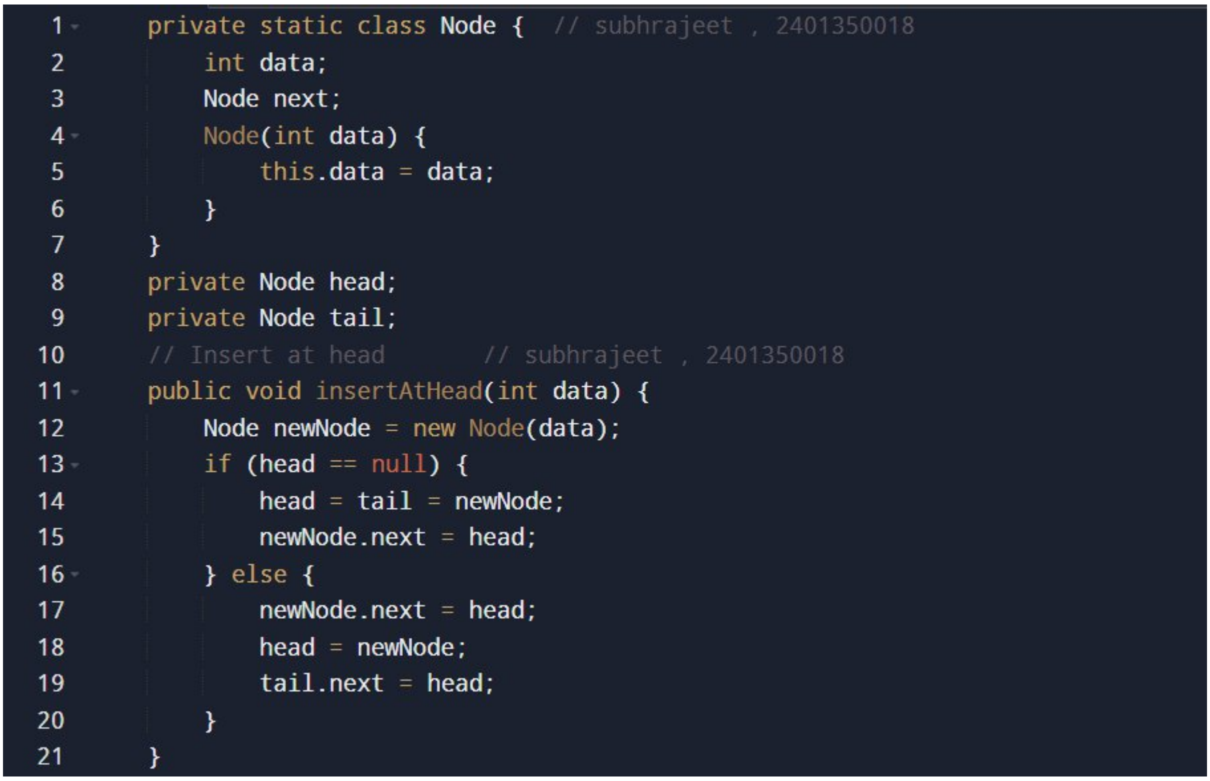
```
        list.insertAtTail(30);
        list.traverse();
        list.deleteByValue(20);
        list.traverse();
    }
}
```

OUTPUT :

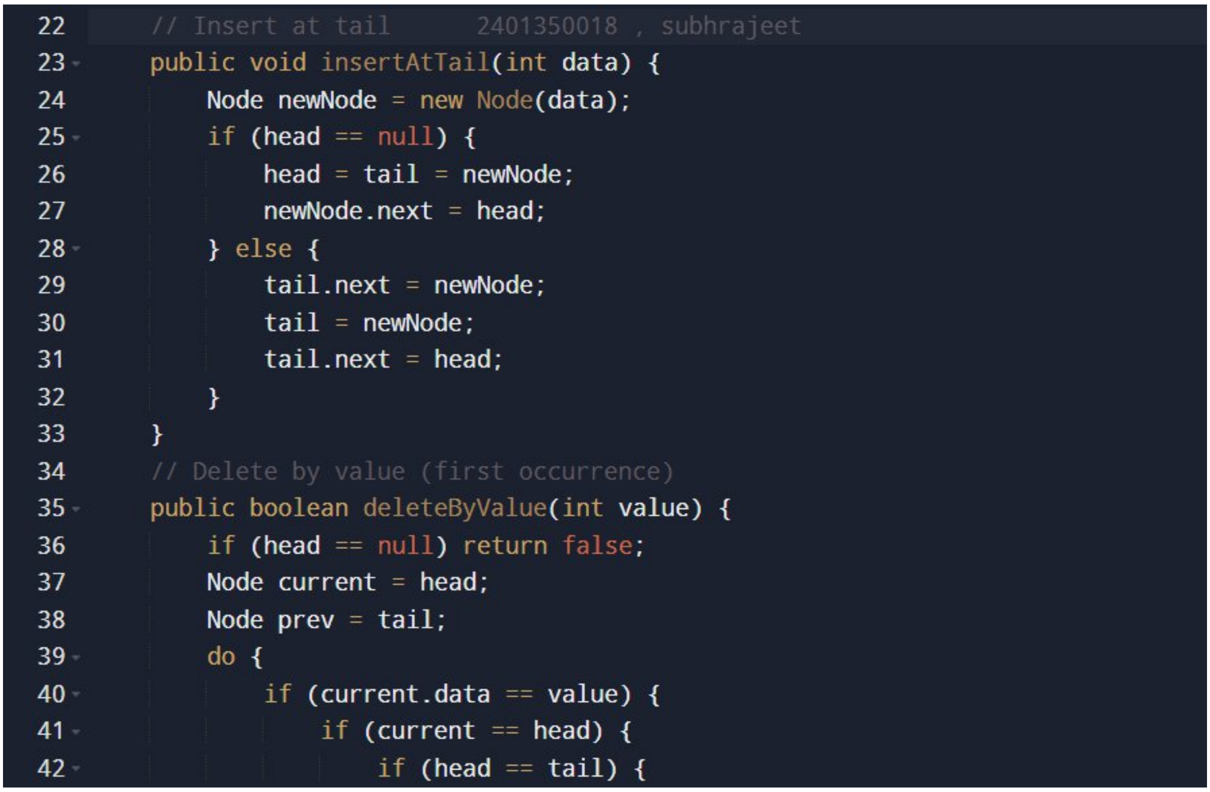


```
PS D:\Subhrajeet_2401350018> javac SinglyLinkedList.java
PS D:\Subhrajeet_2401350018> java SinglyLinkedList
10 20 30
10 30
```

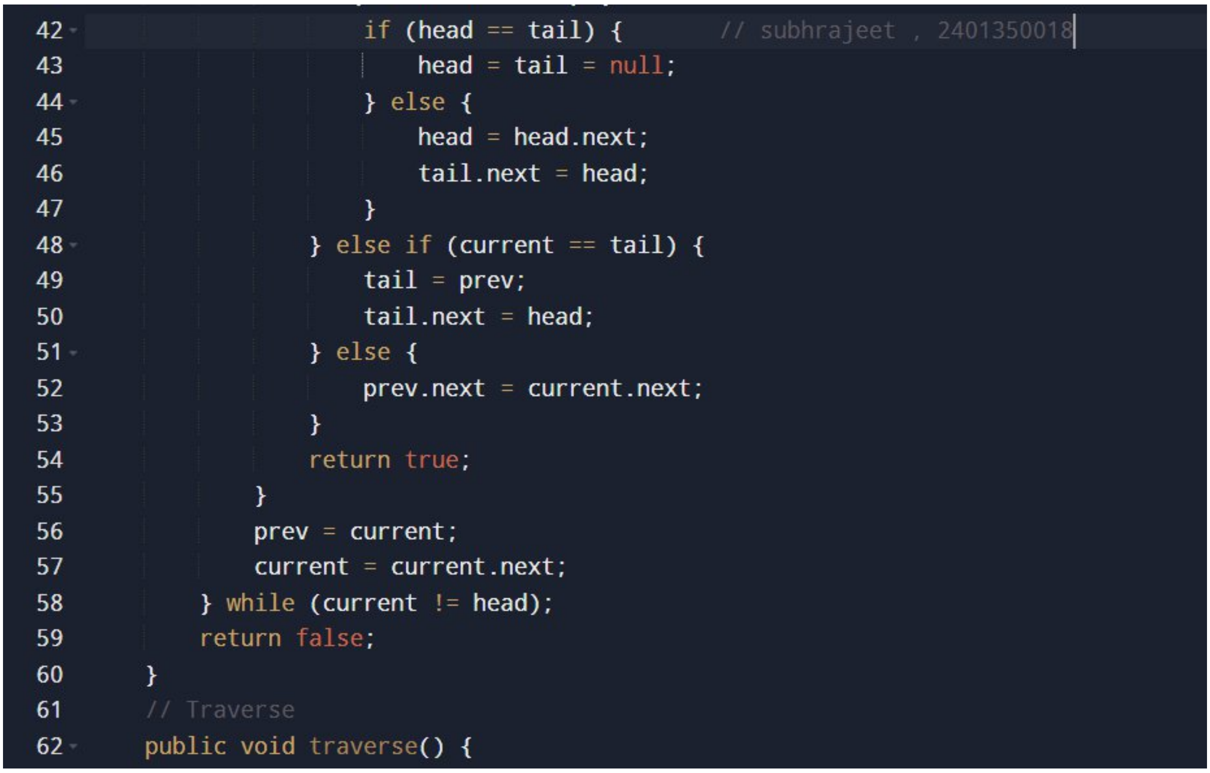
Q3. Write a class to implement a circular linked list with methods to insert an element at the head, insert an element at the tail, delete an element by value, and traverse the list to print all elements.



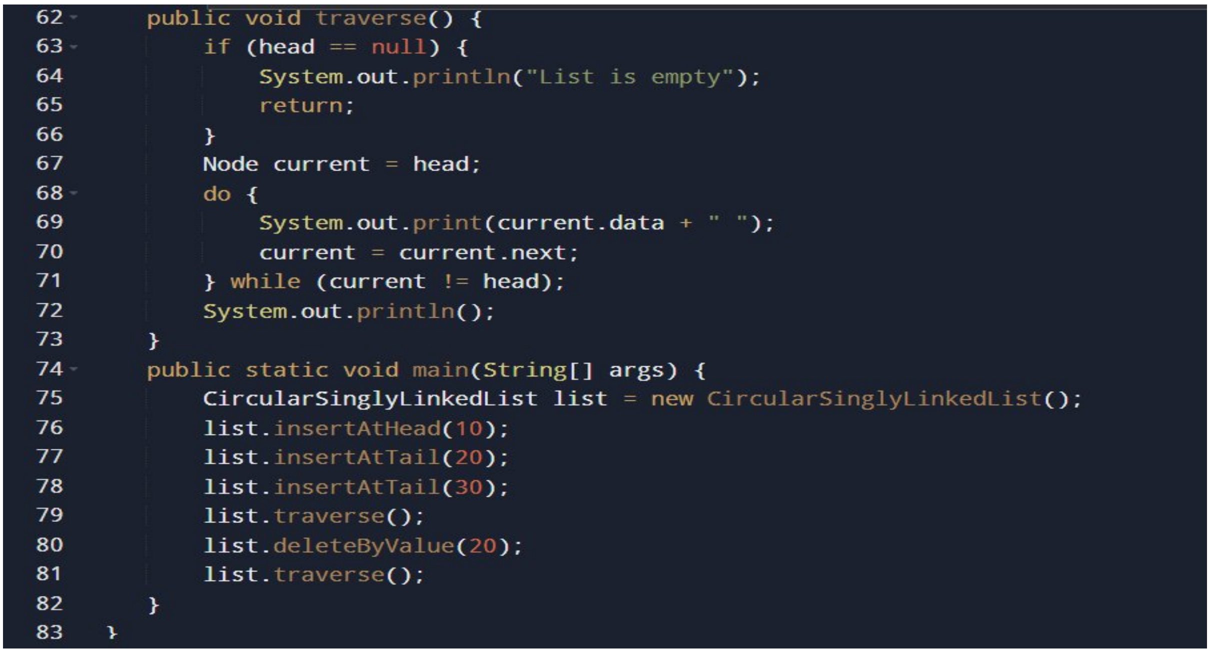
```
private static class Node {  // subhrajeet , 2401350018
    int data;
    Node next;
    Node(int data) {
        this.data = data;
    }
}
private Node head;
private Node tail;
// Insert at head       // subhrajeet , 2401350018
public void insertAtHead(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = tail = newNode;
        newNode.next = head;
    } else {
        newNode.next = head;
        head = newNode;
        tail.next = head;
    }
}
```



```
// Insert at tail      2401350018 , subhrajeet
public void insertAtTail(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = tail = newNode;
        newNode.next = head;
    } else {
        tail.next = newNode;
        tail = newNode;
        tail.next = head;
    }
}
// Delete by value (first occurrence)
public boolean deleteByValue(int value) {
    if (head == null) return false;
    Node current = head;
    Node prev = tail;
    do {
        if (current.data == value) {
            if (current == head) {
                if (head == tail) {
```

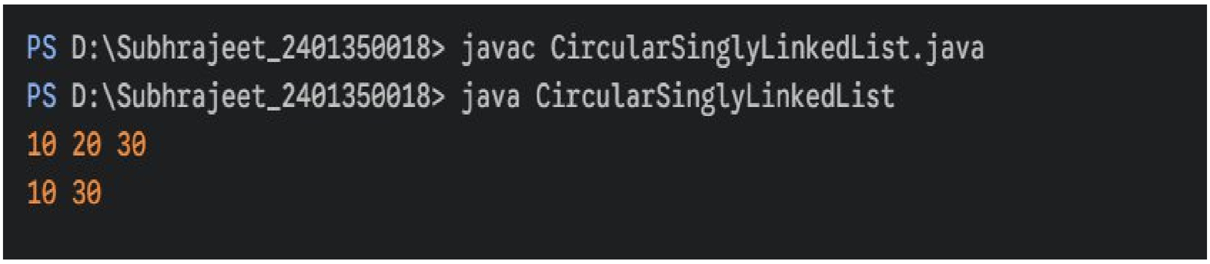


```
                    if (head == tail) {        // subhrajeet , 2401350018
                        head = tail = null;
                    } else {
                        head = head.next;
                        tail.next = head;
                    }
                } else if (current == tail) {
                    tail = prev;
                    tail.next = head;
                } else {
                    prev.next = current.next;
                }
                return true;
            }
            prev = current;
            current = current.next;
        } while (current != head);
        return false;
    }
    // Traverse
    public void traverse() {
```



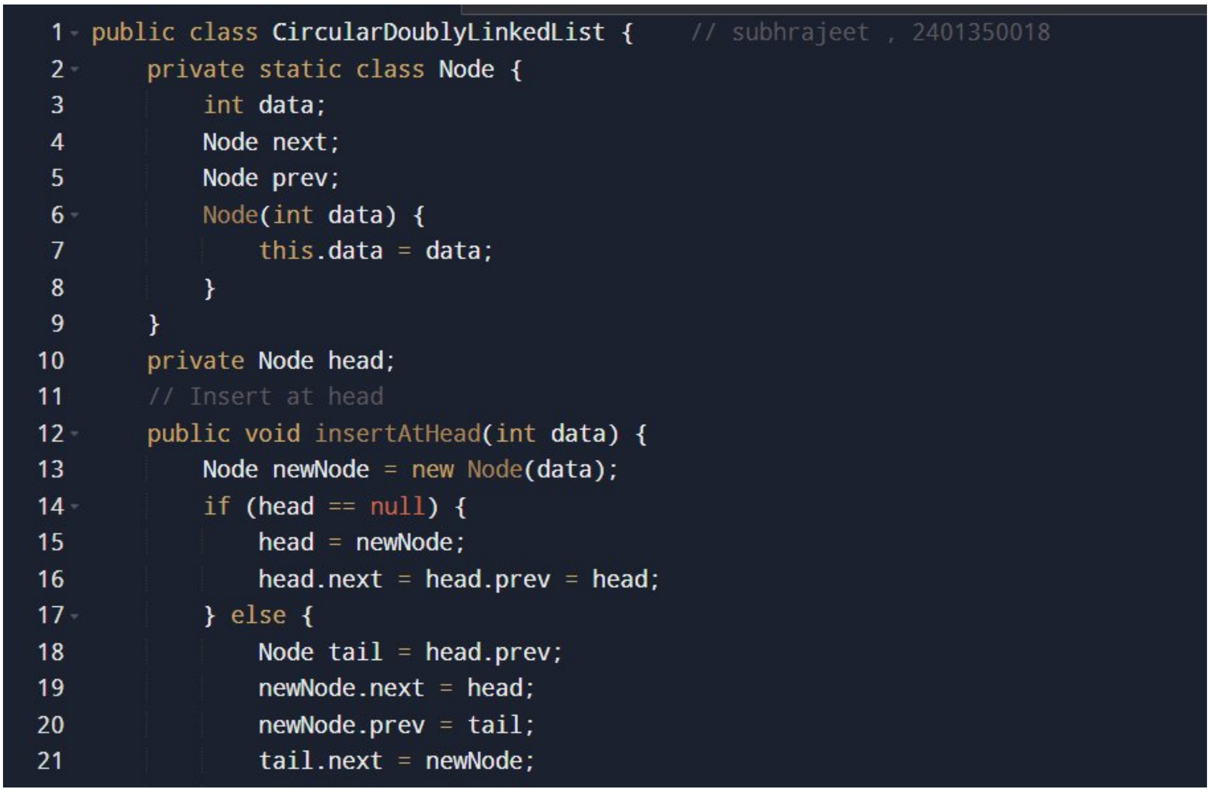
```
    public void traverse() {
        if (head == null) {
            System.out.println("List is empty");
            return;
        }
        Node current = head;
        do {
            System.out.print(current.data + " ");
            current = current.next;
        } while (current != head);
        System.out.println();
    }
    public static void main(String[] args) {
        CircularSinglyLinkedList list = new CircularSinglyLinkedList();
        list.insertAtHead(10);
        list.insertAtTail(20);
        list.insertAtTail(30);
        list.traverse();
        list.deleteByValue(20);
        list.traverse();
    }
}
```

OUTPUT :

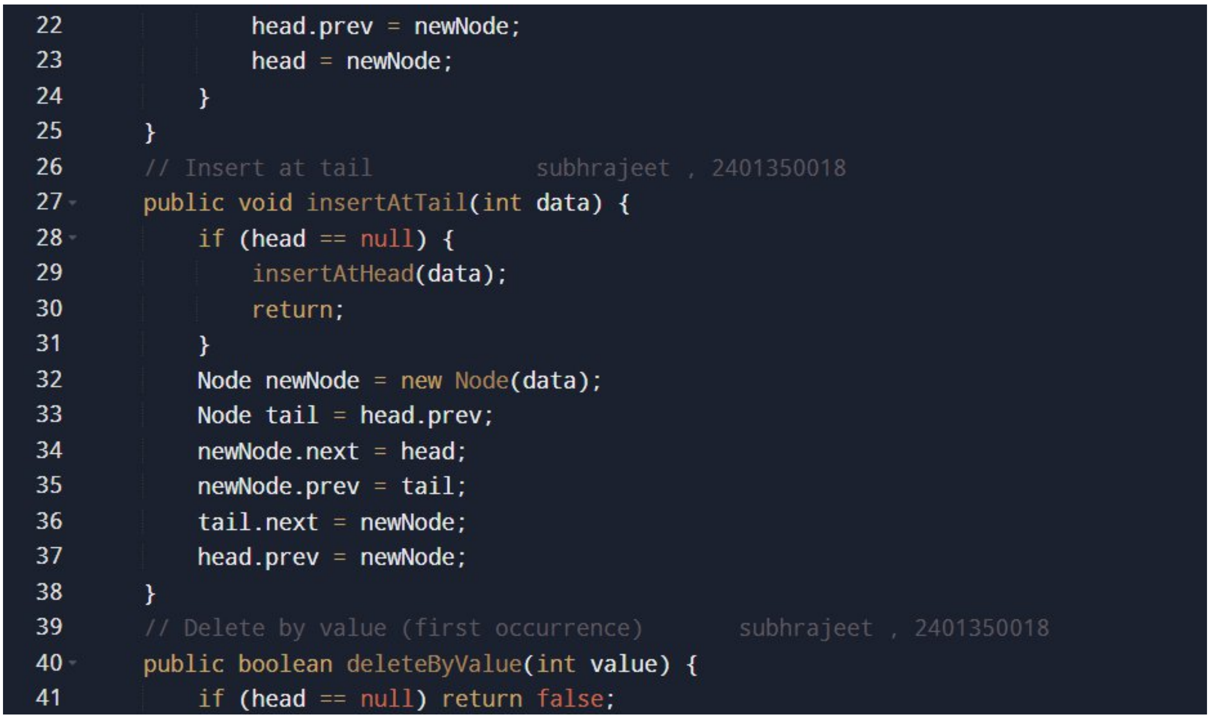


```
PS D:\Subhrajeet_2401350018> javac CircularSinglyLinkedList.java
PS D:\Subhrajeet_2401350018> java CircularSinglyLinkedList
10 20 30
10 30
```

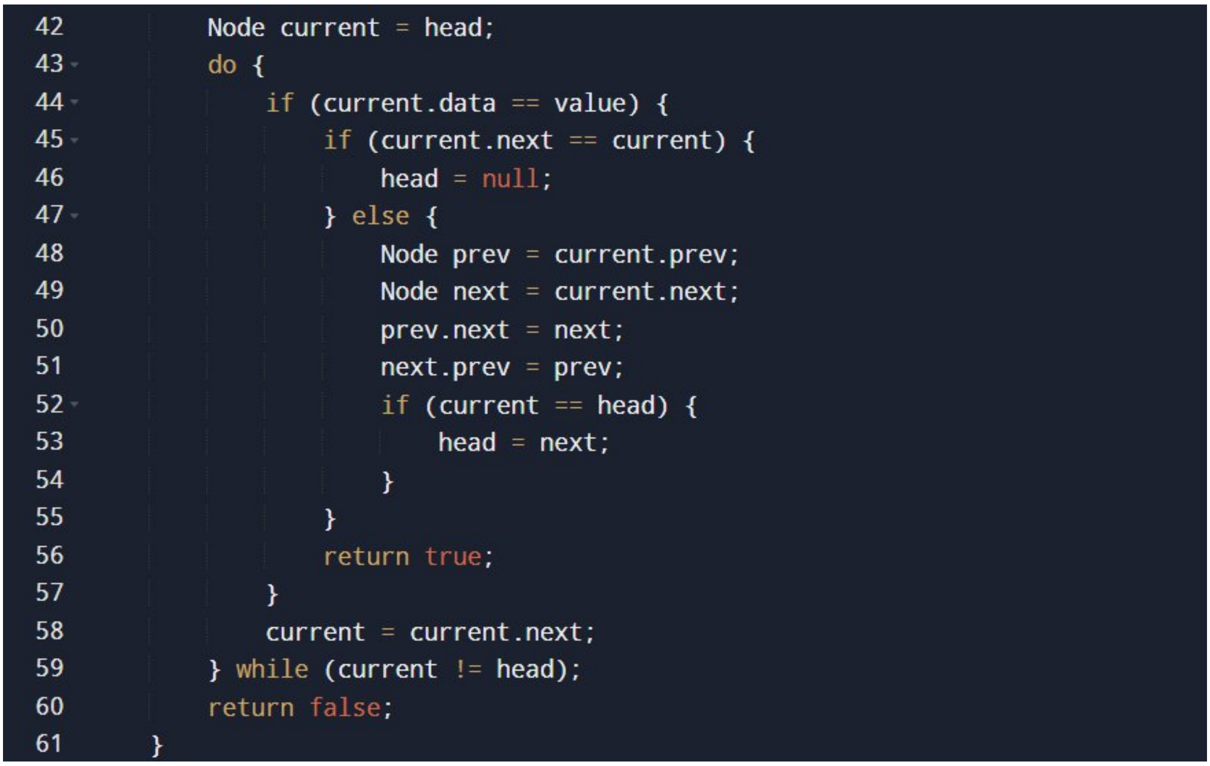
Q4. Implement a doubly linked list with methods to insert an element at the head, insert an element at the tail, delete an element by value, and reverse the list. Ensure that all operations handle edge cases appropriately.



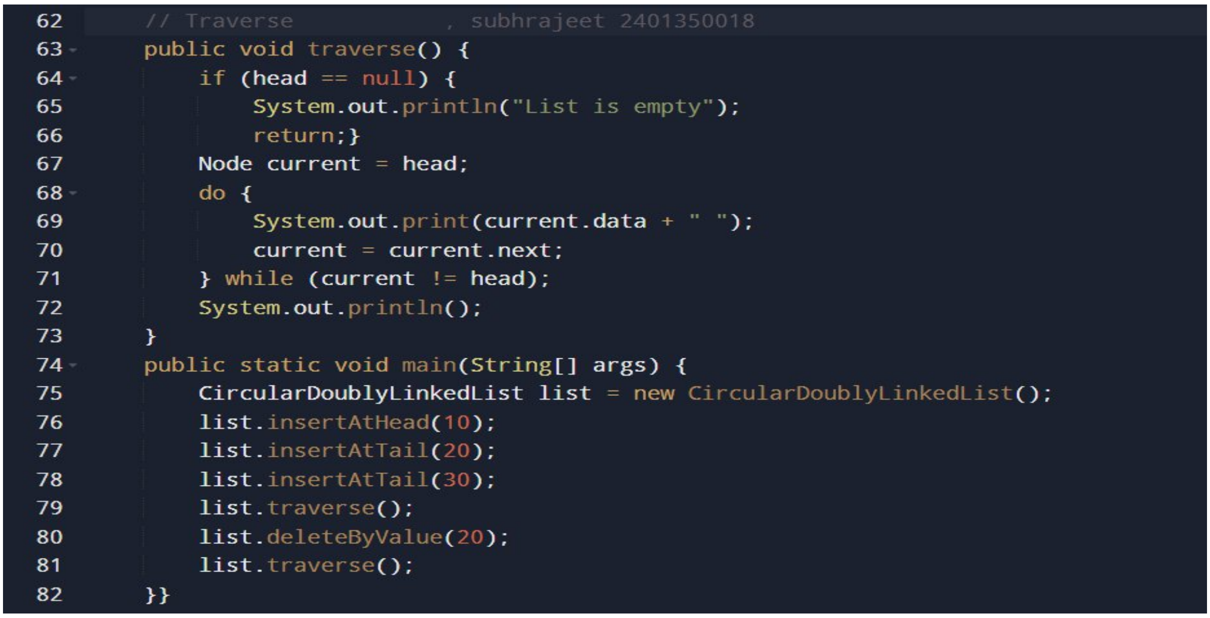
```java
public class CircularDoublyLinkedList {    // subhrajeet , 2401350018
    private static class Node {
        int data;
        Node next;
        Node prev;
        Node(int data) {
            this.data = data;
        }
    }
    private Node head;
    // Insert at head
    public void insertAtHead(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            head.next = head.prev = head;
        } else {
            Node tail = head.prev;
            newNode.next = head;
            newNode.prev = tail;
            tail.next = newNode;
```



```java
            head.prev = newNode;
            head = newNode;
        }
    }
    // Insert at tail              subhrajeet , 2401350018
    public void insertAtTail(int data) {
        if (head == null) {
            insertAtHead(data);
            return;
        }
        Node newNode = new Node(data);
        Node tail = head.prev;
        newNode.next = head;
        newNode.prev = tail;
        tail.next = newNode;
        head.prev = newNode;
    }
    // Delete by value (first occurrence)        subhrajeet , 2401350018
    public boolean deleteByValue(int value) {
        if (head == null) return false;
```

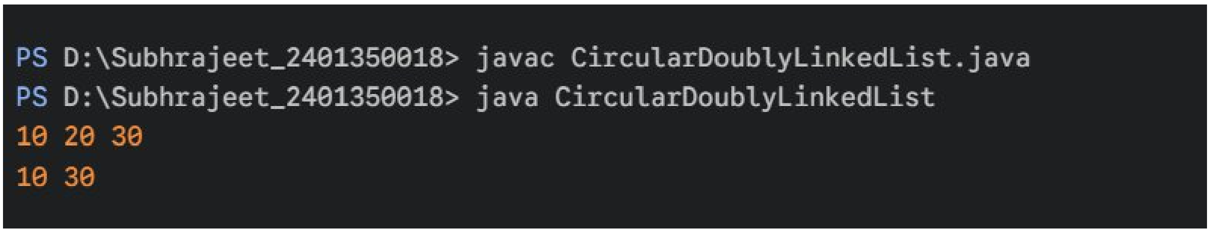


```java
        Node current = head;
        do {
            if (current.data == value) {
                if (current.next == current) {
                    head = null;
                } else {
                    Node prev = current.prev;
                    Node next = current.next;
                    prev.next = next;
                    next.prev = prev;
                    if (current == head) {
                        head = next;
                    }
                }
                return true;
            }
            current = current.next;
        } while (current != head);
        return false;
    }
```



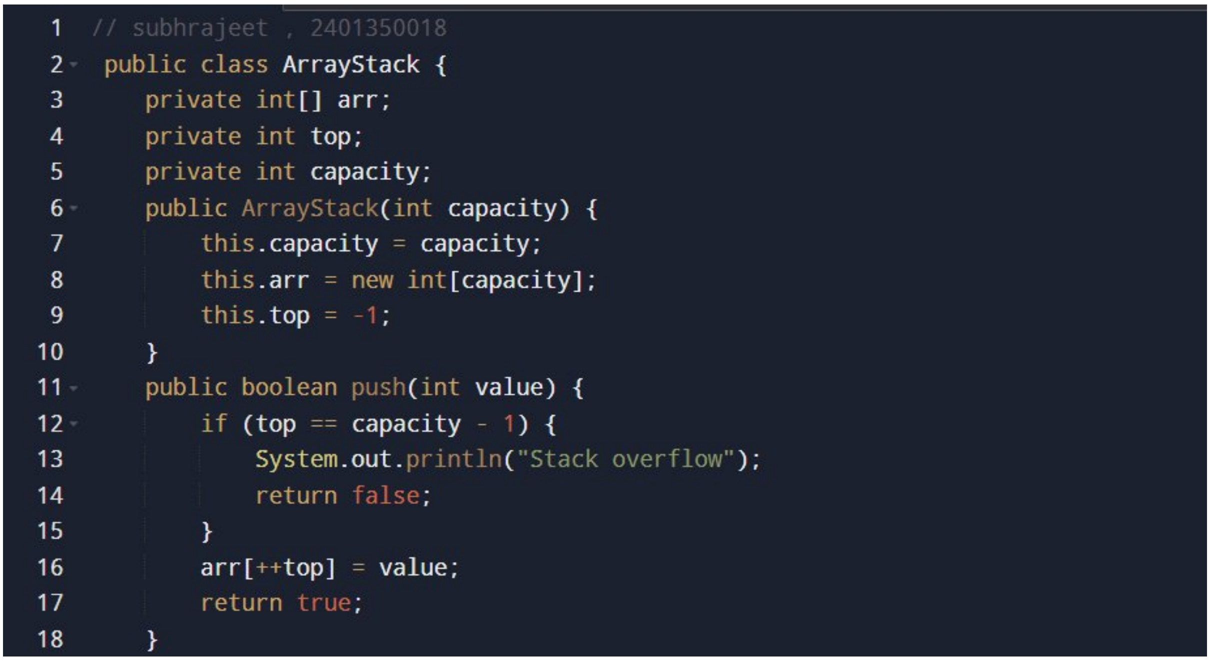
```java
    // Traverse               , subhrajeet 2401350018
    public void traverse() {
        if (head == null) {
            System.out.println("List is empty");
            return;}
        Node current = head;
        do {
            System.out.print(current.data + " ");
            current = current.next;
        } while (current != head);
        System.out.println();
    }
    public static void main(String[] args) {
        CircularDoublyLinkedList list = new CircularDoublyLinkedList();
        list.insertAtHead(10);
        list.insertAtTail(20);
        list.insertAtTail(30);
        list.traverse();
        list.deleteByValue(20);
        list.traverse();
    }}
```

OUTPUT :

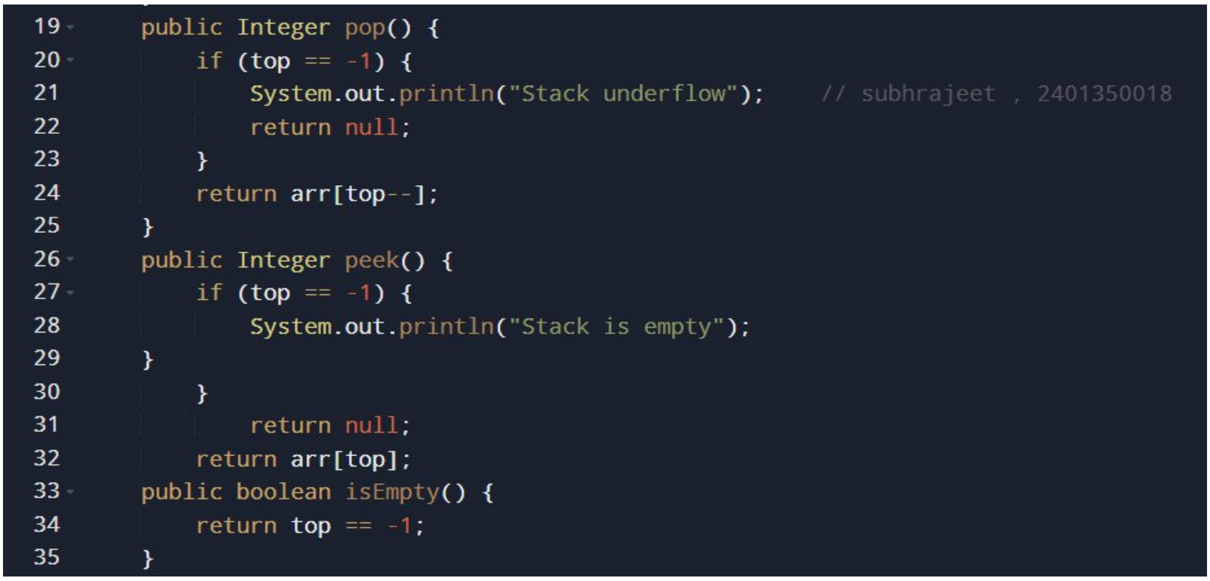


```
PS D:\Subhrajeet_2401350018> javac CircularDoublyLinkedList.java
PS D:\Subhrajeet_2401350018> java CircularDoublyLinkedList
10 20 30
10 30
```

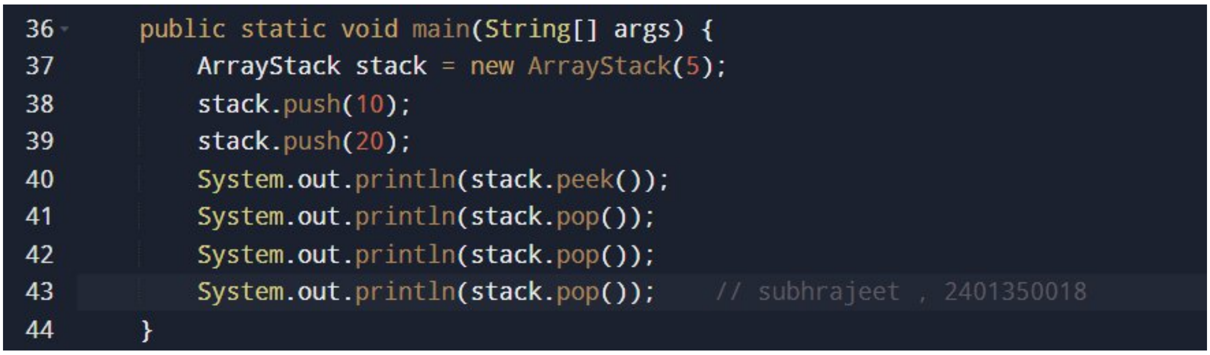
Q5. Implement a stack using arrays with methods for push, pop, and peek operations.



```java
// subhrajeet , 2401350018
public class ArrayStack {
    private int[] arr;
    private int top;
    private int capacity;
    public ArrayStack(int capacity) {
        this.capacity = capacity;
        this.arr = new int[capacity];
        this.top = -1;
    }
    public boolean push(int value) {
        if (top == capacity - 1) {
            System.out.println("Stack overflow");
            return false;
        }
        arr[++top] = value;
        return true;
    }
```

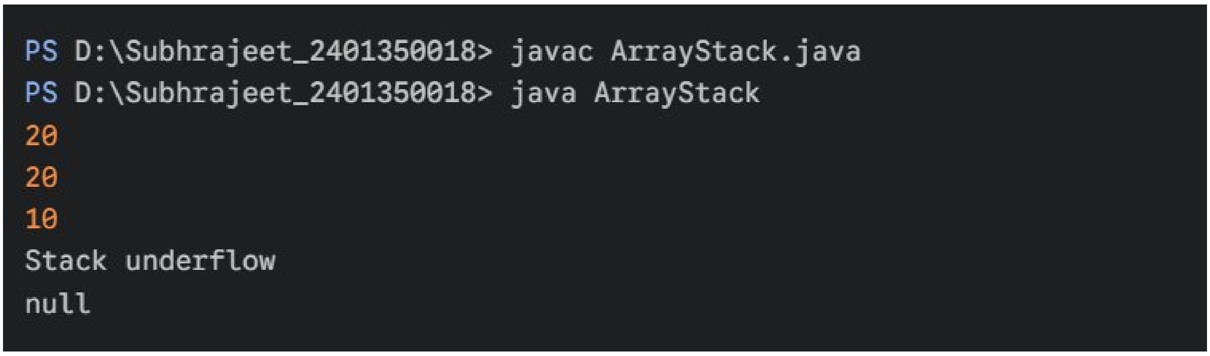


```java
    public Integer pop() {
        if (top == -1) {
            System.out.println("Stack underflow");    // subhrajeet , 2401350018
            return null;
        }
        return arr[top--];
    }
    public Integer peek() {
        if (top == -1) {
            System.out.println("Stack is empty");
    }
        }
            return null;
        return arr[top];
    public boolean isEmpty() {
        return top == -1;
    }
```



```java
    public static void main(String[] args) {
        ArrayStack stack = new ArrayStack(5);
        stack.push(10);
        stack.push(20);
        System.out.println(stack.peek());
        System.out.println(stack.pop());
        System.out.println(stack.pop());
        System.out.println(stack.pop());    // subhrajeet , 2401350018
    }
```

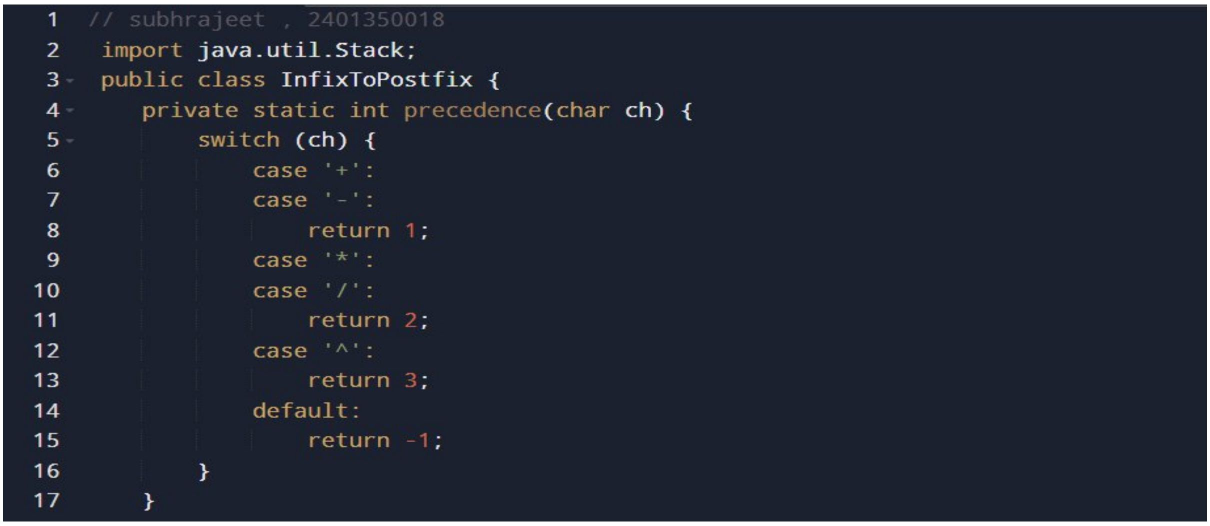
OUTPUT :



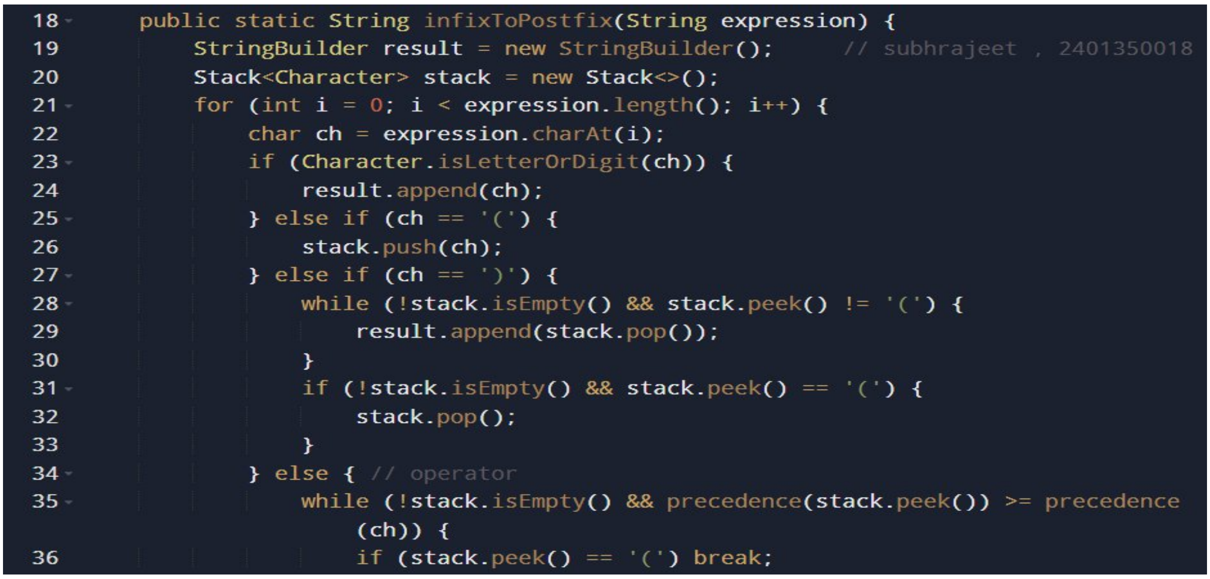
```
PS D:\Subhrajeet_2401350018> javac ArrayStack.java
PS D:\Subhrajeet_2401350018> java ArrayStack
20
20
10
Stack underflow
null
```

Q6. Write a function to convert an infix expression to a postfix expression using a stack. The function should handle parentheses and operator precedence correctly.

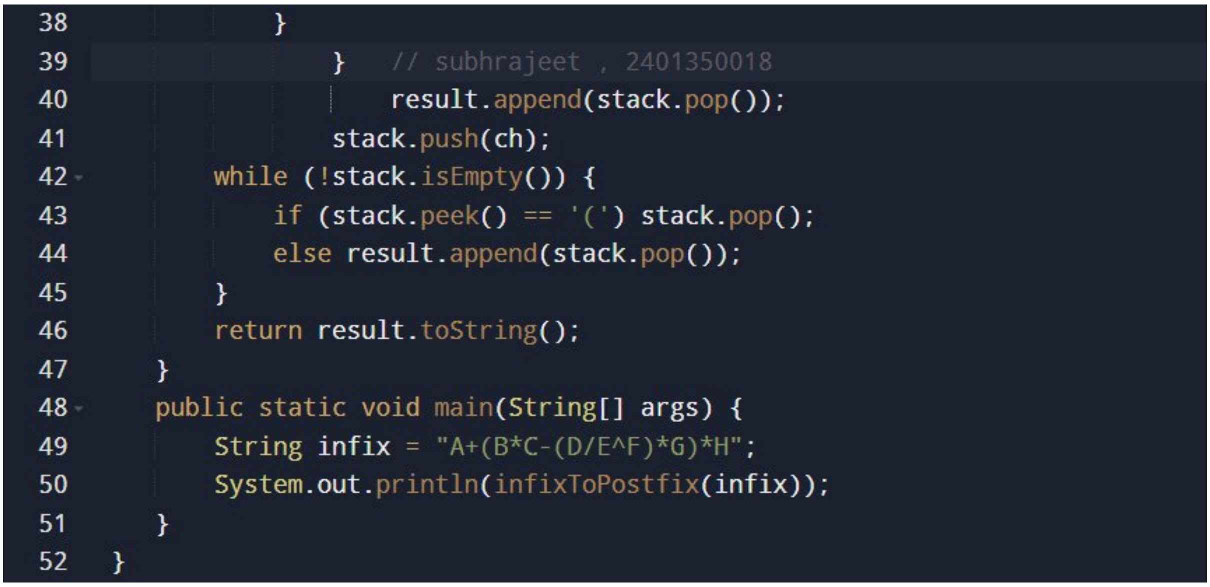
CODE :



```java
// subhrajeet , 2401350018
import java.util.Stack;
public class InfixToPostfix {
    private static int precedence(char ch) {
        switch (ch) {
            case '+':
            case '-':
                return 1;
            case '*':
            case '/':
                return 2;
            case '^':
                return 3;
            default:
                return -1;
        }
    }
```

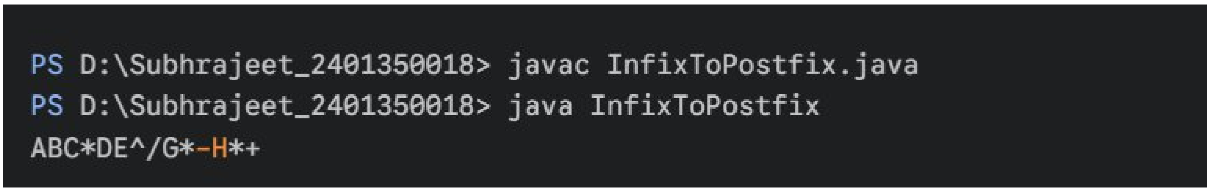


```
    public static String infixToPostfix(String expression) {
        StringBuilder result = new StringBuilder();      // subhrajeet , 2401350018
        Stack<Character> stack = new Stack<>();
        for (int i = 0; i < expression.length(); i++) {
            char ch = expression.charAt(i);
            if (Character.isLetterOrDigit(ch)) {
                result.append(ch);
            } else if (ch == '(') {
                stack.push(ch);
            } else if (ch == ')') {
                while (!stack.isEmpty() && stack.peek() != '(') {
                    result.append(stack.pop());
                }
                if (!stack.isEmpty() && stack.peek() == '(') {
                    stack.pop();
                }
            } else { // operator
                while (!stack.isEmpty() && precedence(stack.peek()) >= precedence
                    (ch)) {
                    if (stack.peek() == '(') break;
```



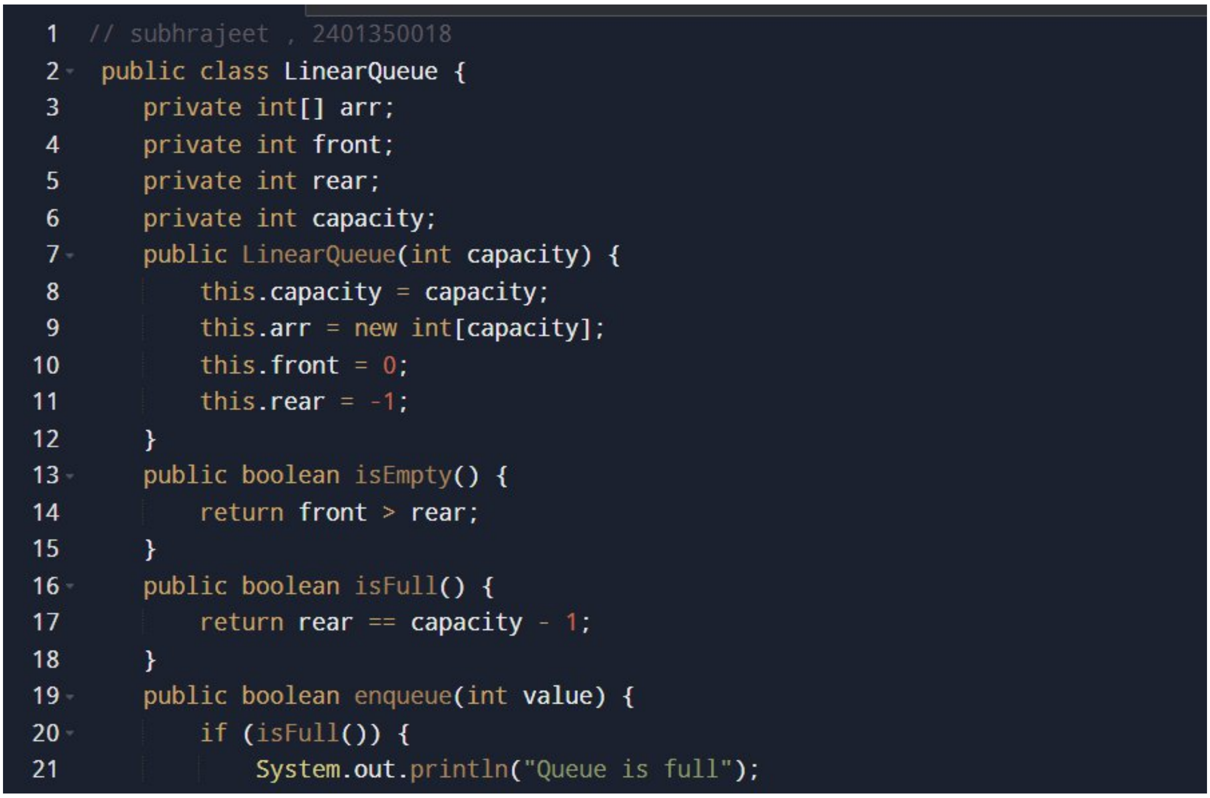
```
            }
                }   // subhrajeet , 2401350018
                    result.append(stack.pop());
                stack.push(ch);
        while (!stack.isEmpty()) {
            if (stack.peek() == '(') stack.pop();
            else result.append(stack.pop());
        }
        return result.toString();
    }
    public static void main(String[] args) {
        String infix = "A+(B*C-(D/E^F)*G)*H";
        System.out.println(infixToPostfix(infix));
    }
}
```

OUTPUT :

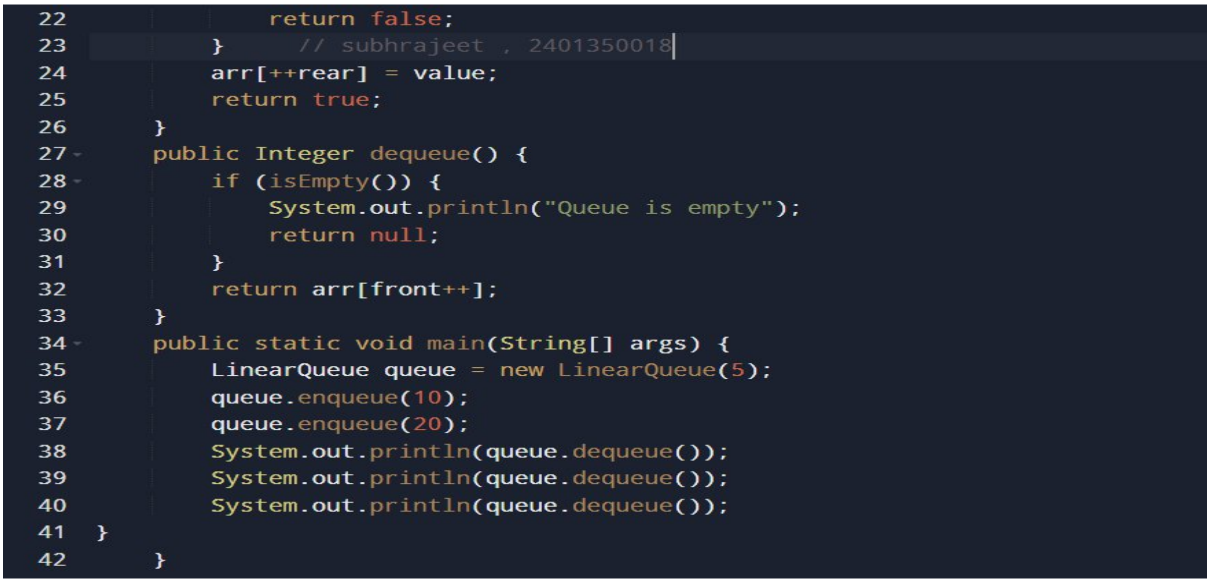


```
PS D:\Subhrajeet_2401350018> javac InfixToPostfix.java
PS D:\Subhrajeet_2401350018> java InfixToPostfix
ABC*DE^/G*-H*+
```

Q7. Create a linear queue using an array with methods for enqueue, dequeue, and checking if the queue is empty or full.

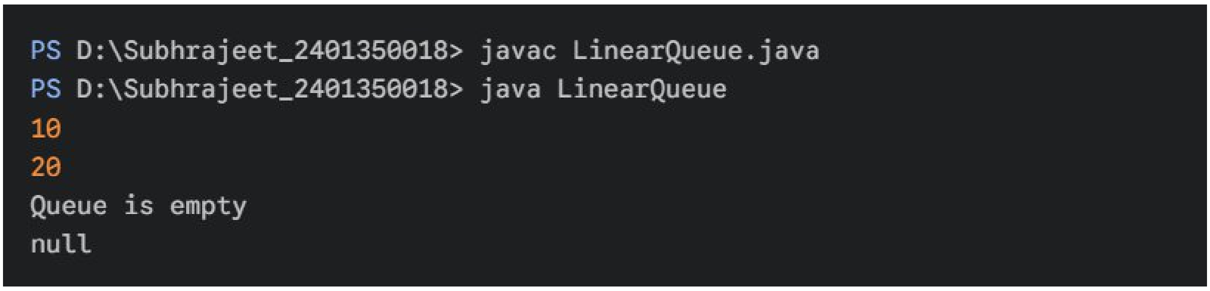


```java
// subhrajeet , 2401350018
public class LinearQueue {
    private int[] arr;
    private int front;
    private int rear;
    private int capacity;
    public LinearQueue(int capacity) {
        this.capacity = capacity;
        this.arr = new int[capacity];
        this.front = 0;
        this.rear = -1;
    }
    public boolean isEmpty() {
        return front > rear;
    }
    public boolean isFull() {
        return rear == capacity - 1;
    }
    public boolean enqueue(int value) {
        if (isFull()) {
            System.out.println("Queue is full");
```



```java
            return false;
        }    // subhrajeet , 2401350018
        arr[++rear] = value;
        return true;
    }
    public Integer dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return null;
        }
        return arr[front++];
    }
    public static void main(String[] args) {
        LinearQueue queue = new LinearQueue(5);
        queue.enqueue(10);
        queue.enqueue(20);
        System.out.println(queue.dequeue());
        System.out.println(queue.dequeue());
        System.out.println(queue.dequeue());
}
    }
```

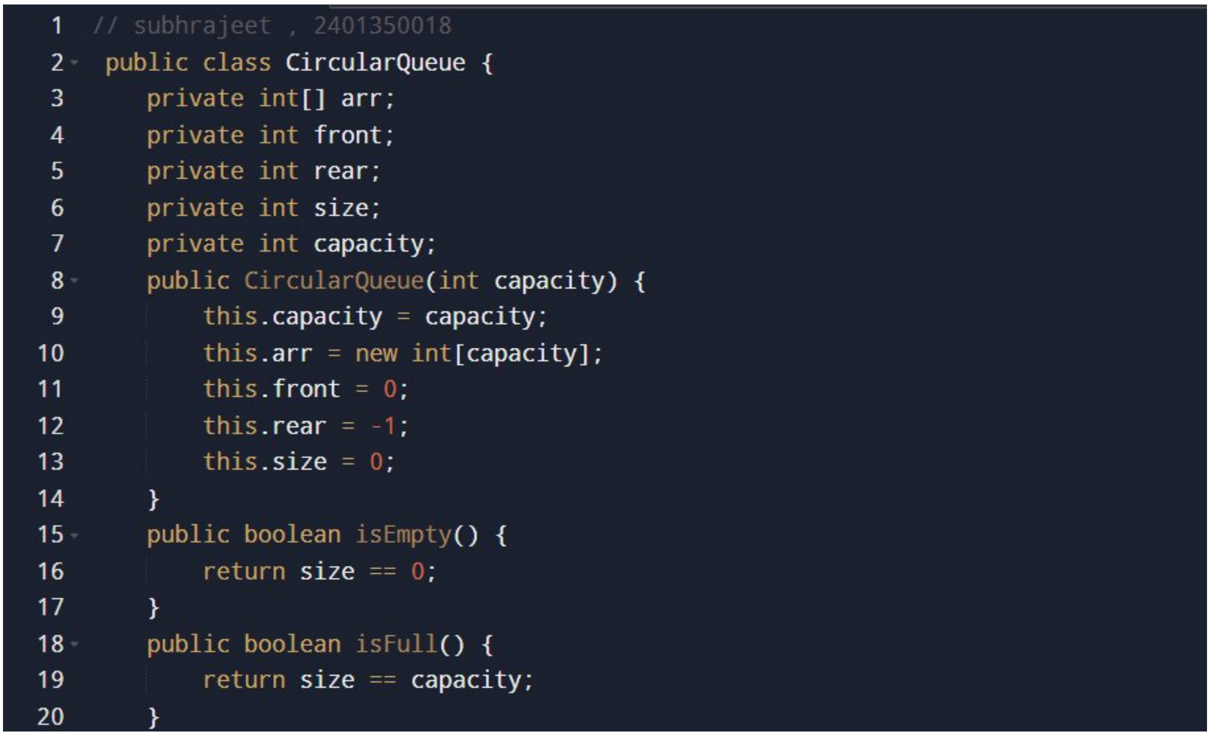
OUTPUT :



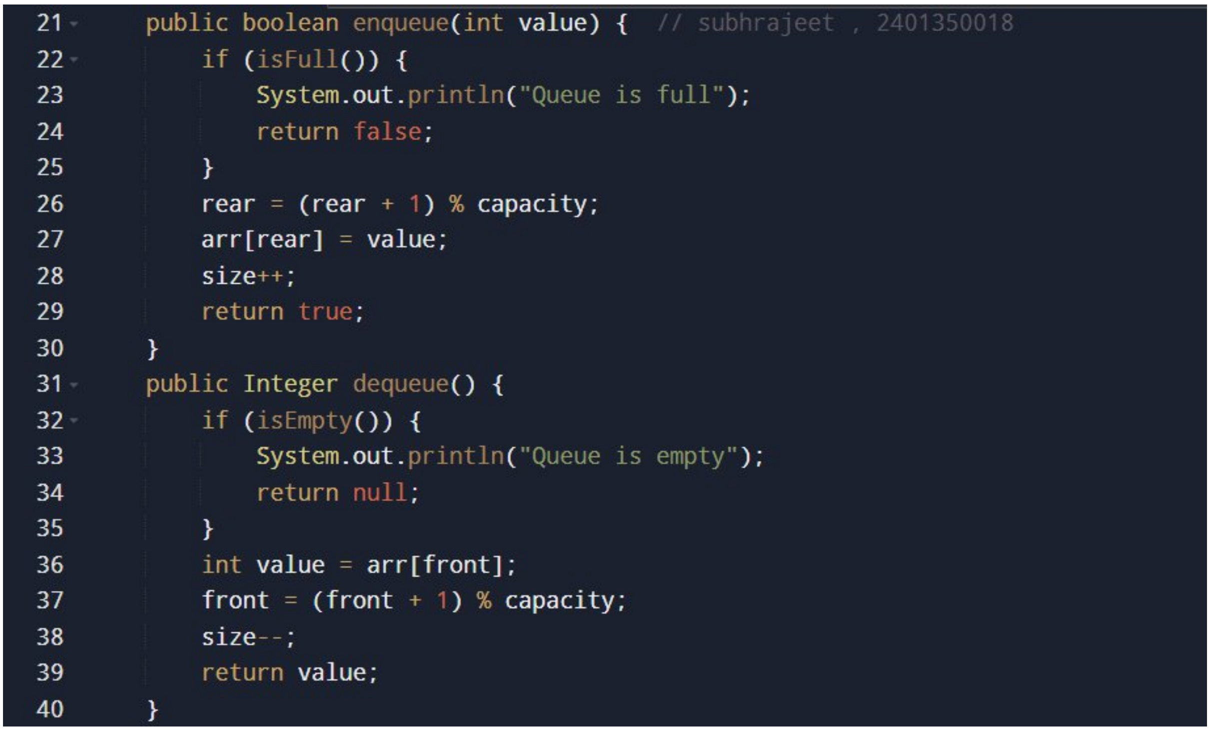
```
PS D:\Subhrajeet_2401350018> javac LinearQueue.java
PS D:\Subhrajeet_2401350018> java LinearQueue
10
20
Queue is empty
null
```

Q8. 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.

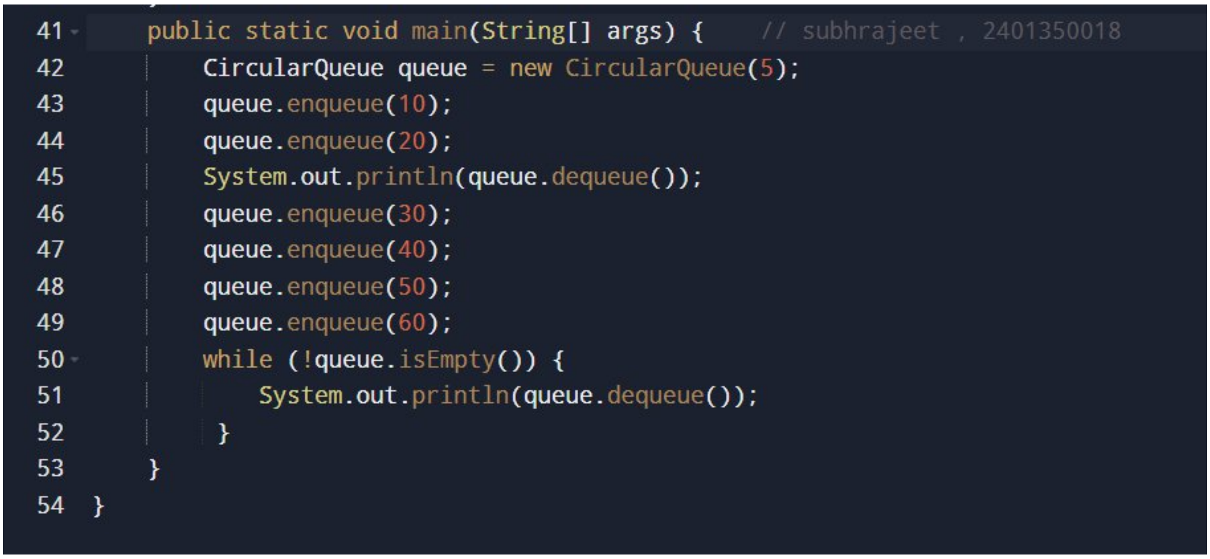
CODE :



```java
// subhrajeet , 2401350018
public class CircularQueue {
    private int[] arr;
    private int front;
    private int rear;
    private int size;
    private int capacity;
    public CircularQueue(int capacity) {
        this.capacity = capacity;
        this.arr = new int[capacity];
        this.front = 0;
        this.rear = -1;
        this.size = 0;
    }
    public boolean isEmpty() {
        return size == 0;
    }
    public boolean isFull() {
        return size == capacity;
    }
```

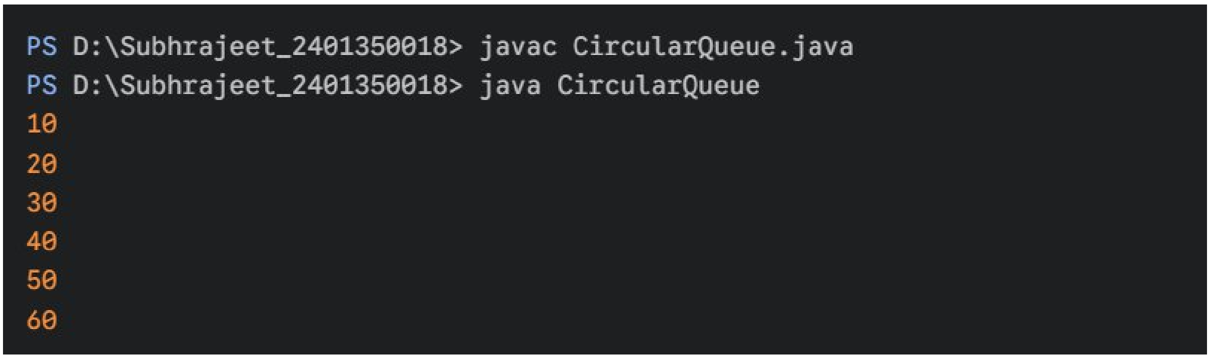


```java
    public boolean enqueue(int value) {  // subhrajeet , 2401350018
        if (isFull()) {
            System.out.println("Queue is full");
            return false;
        }
        rear = (rear + 1) % capacity;
        arr[rear] = value;
        size++;
        return true;
    }
    public Integer dequeue() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return null;
        }
        int value = arr[front];
        front = (front + 1) % capacity;
        size--;
        return value;
    }
```



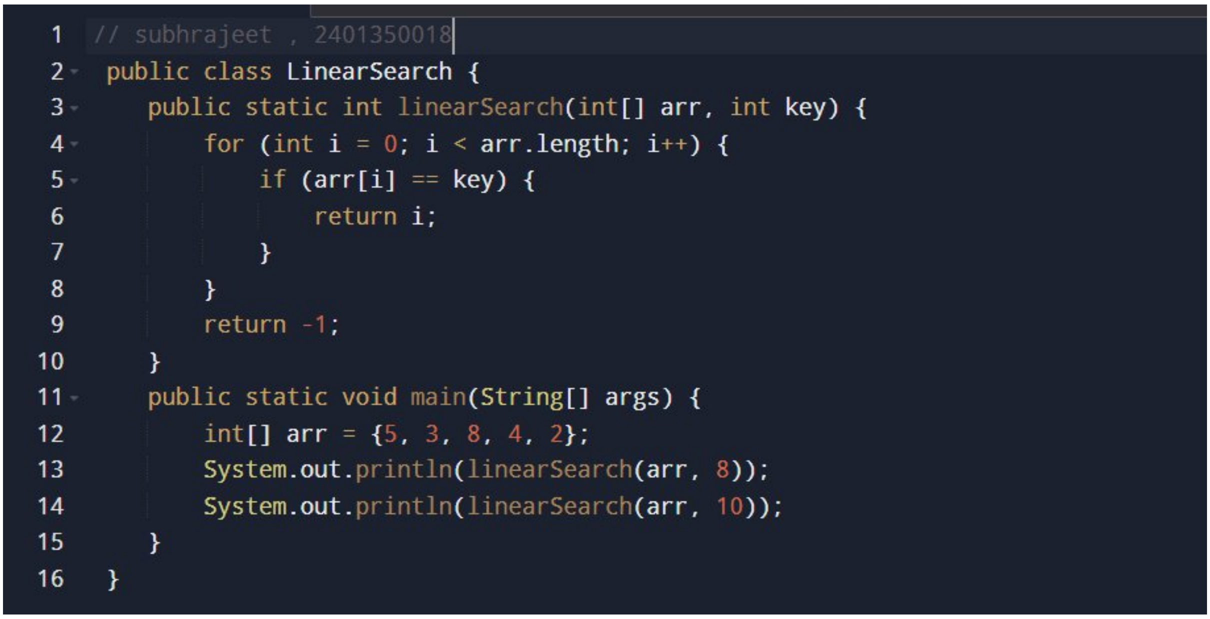
```java
    public static void main(String[] args) {    // subhrajeet , 2401350018
        CircularQueue queue = new CircularQueue(5);
        queue.enqueue(10);
        queue.enqueue(20);
        System.out.println(queue.dequeue());
        queue.enqueue(30);
        queue.enqueue(40);
        queue.enqueue(50);
        queue.enqueue(60);
        while (!queue.isEmpty()) {
            System.out.println(queue.dequeue());
        }
    }
}
```

OUTPUT :



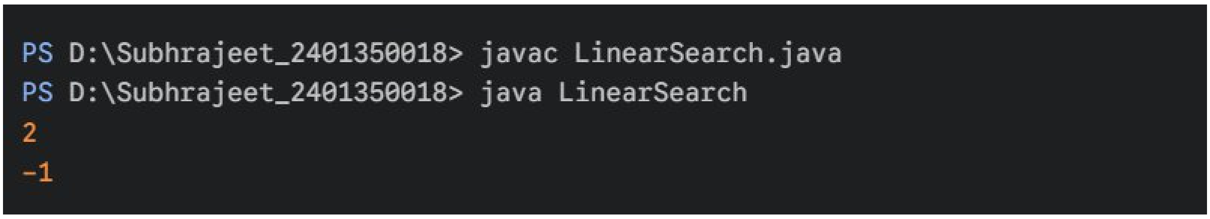
```
PS D:\Subhrajeet_2401350018> javac CircularQueue.java
PS D:\Subhrajeet_2401350018> java CircularQueue
10
20
30
40
50
60
```

Q9. Implement linear search



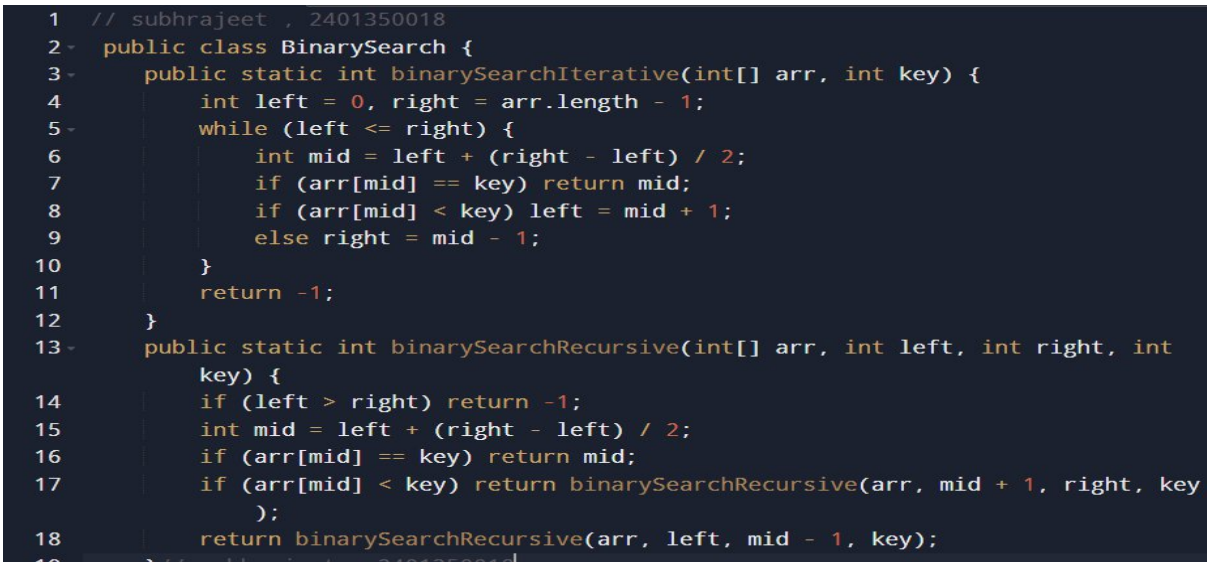
```java
// subhrajeet , 2401350018
public class LinearSearch {
    public static int linearSearch(int[] arr, int key) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == key) {
                return i;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        int[] arr = {5, 3, 8, 4, 2};
        System.out.println(linearSearch(arr, 8));
        System.out.println(linearSearch(arr, 10));
    }
}
```

OUTPUT :

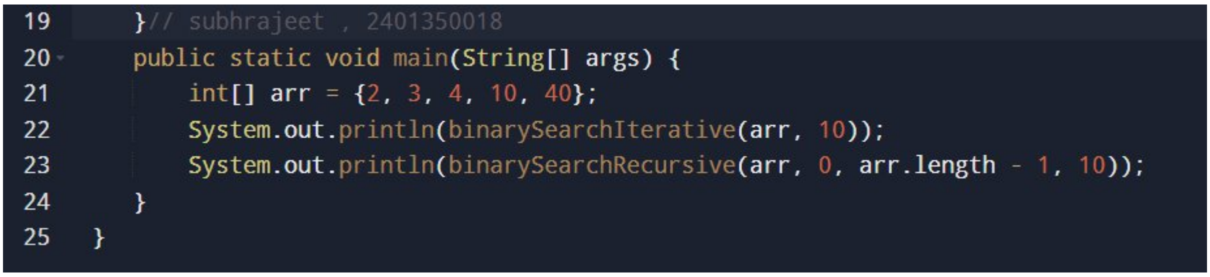


```
PS D:\Subhrajeet_2401350018> javac LinearSearch.java
PS D:\Subhrajeet_2401350018> java LinearSearch
2
-1
```

Q10. Implement binary search(iterative and recursive)

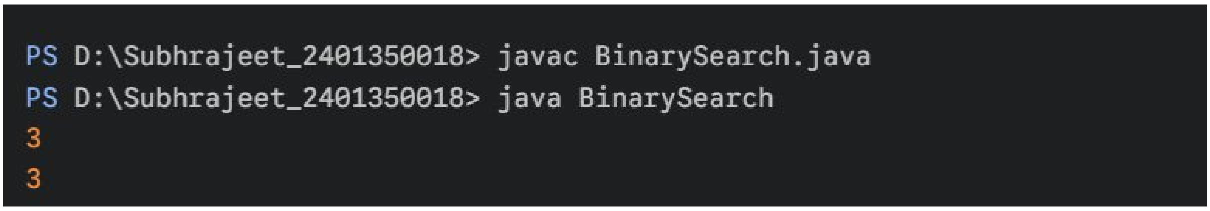


```java
// subhrajeet , 2401350018
public class BinarySearch {
    public static int binarySearchIterative(int[] arr, int key) {
        int left = 0, right = arr.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (arr[mid] == key) return mid;
            if (arr[mid] < key) left = mid + 1;
            else right = mid - 1;
        }
        return -1;
    }
    public static int binarySearchRecursive(int[] arr, int left, int right, int key) {
        if (left > right) return -1;
        int mid = left + (right - left) / 2;
        if (arr[mid] == key) return mid;
        if (arr[mid] < key) return binarySearchRecursive(arr, mid + 1, right, key);
        return binarySearchRecursive(arr, left, mid - 1, key);
```



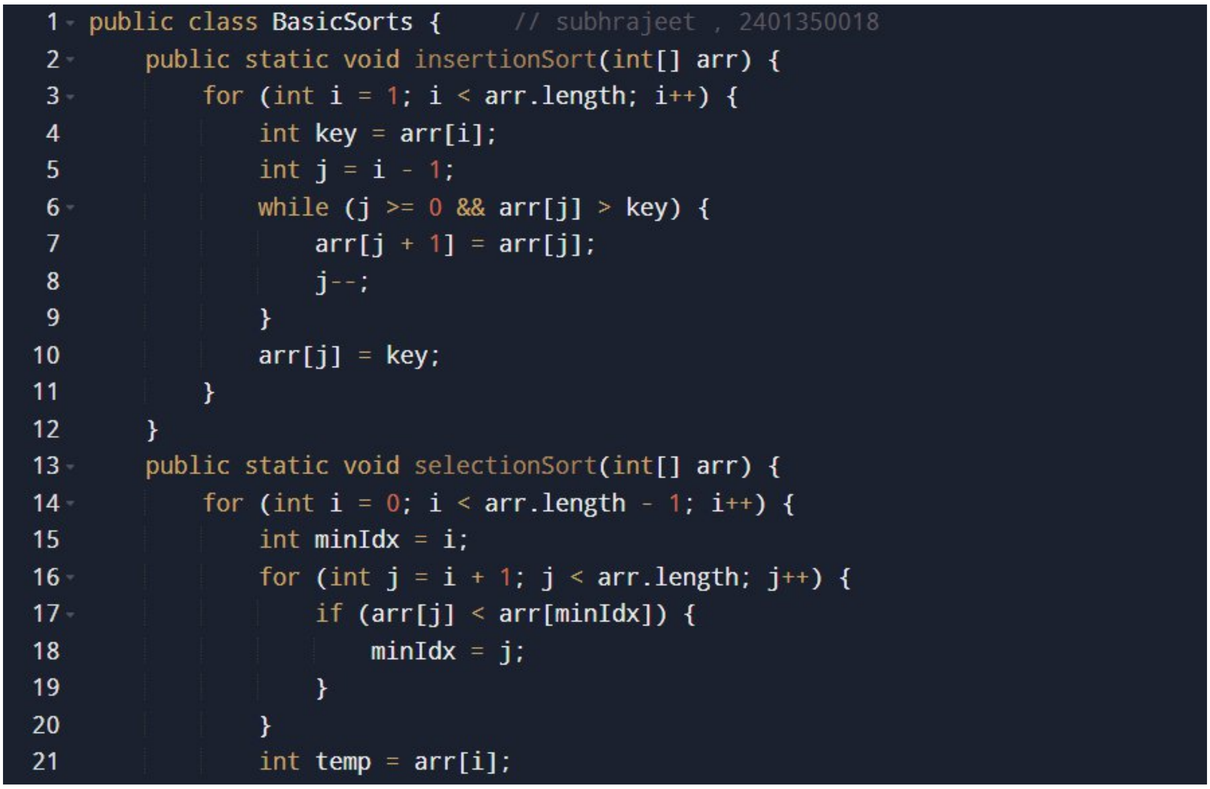
```java
    }// subhrajeet , 2401350018
    public static void main(String[] args) {
        int[] arr = {2, 3, 4, 10, 40};
        System.out.println(binarySearchIterative(arr, 10));
        System.out.println(binarySearchRecursive(arr, 0, arr.length - 1, 10));
    }
}
```

OUTPUT :

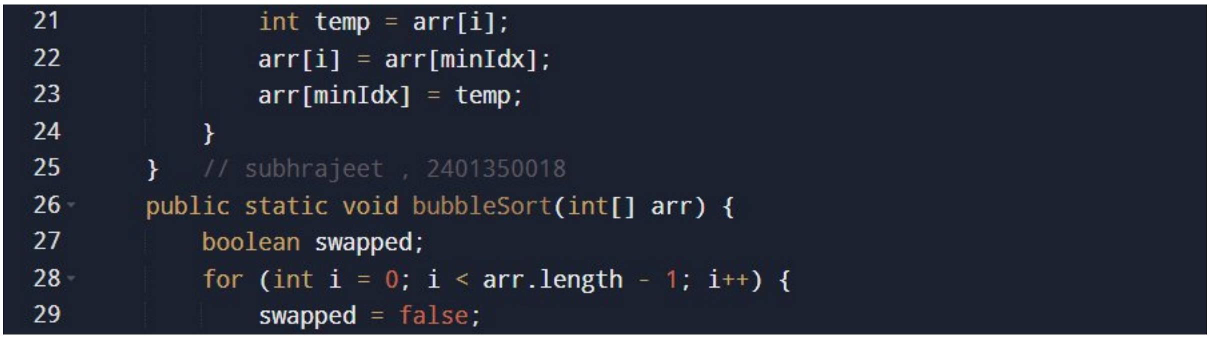


```
PS D:\Subhrajeet_2401350018> javac BinarySearch.java
PS D:\Subhrajeet_2401350018> java BinarySearch
3
3
```

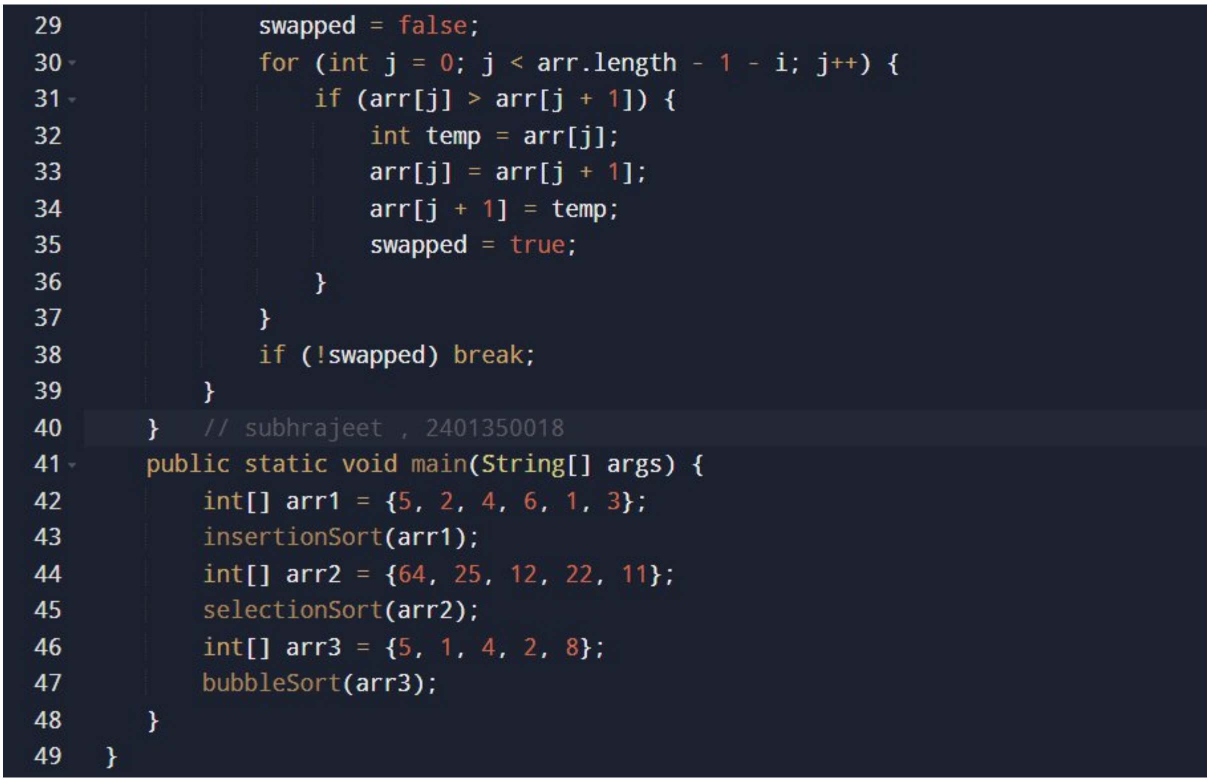
Q11 . Implement various sorting algorithms including Insertion sort, selection sort, bubble sort, and analyze their performance on different input sizes.



```java
public class BasicSorts {     // subhrajeet , 2401350018
    public static void insertionSort(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j] = key;
        }
    }
    public static void selectionSort(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            int minIdx = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[minIdx]) {
                    minIdx = j;
                }
            }
            int temp = arr[i];
```

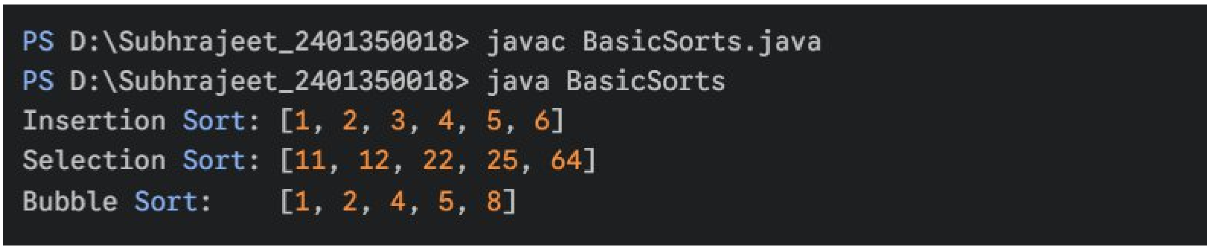


```
            int temp = arr[i];
            arr[i] = arr[minIdx];
            arr[minIdx] = temp;
        }
    }   // subhrajeet , 2401350018
    public static void bubbleSort(int[] arr) {
        boolean swapped;
        for (int i = 0; i < arr.length - 1; i++) {
            swapped = false;
```



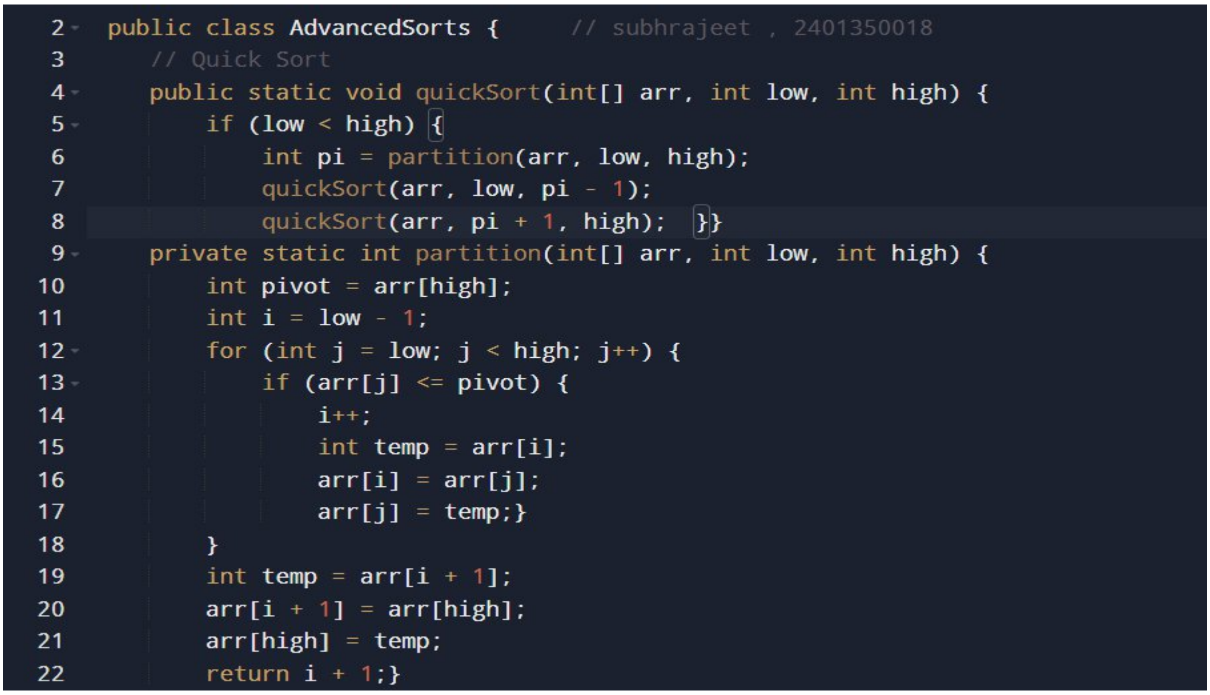
```
            swapped = false;
            for (int j = 0; j < arr.length - 1 - i; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }
            if (!swapped) break;
        }
    }   // subhrajeet , 2401350018
    public static void main(String[] args) {
        int[] arr1 = {5, 2, 4, 6, 1, 3};
        insertionSort(arr1);
        int[] arr2 = {64, 25, 12, 22, 11};
        selectionSort(arr2);
        int[] arr3 = {5, 1, 4, 2, 8};
        bubbleSort(arr3);
    }
}
```

OUTPUT :

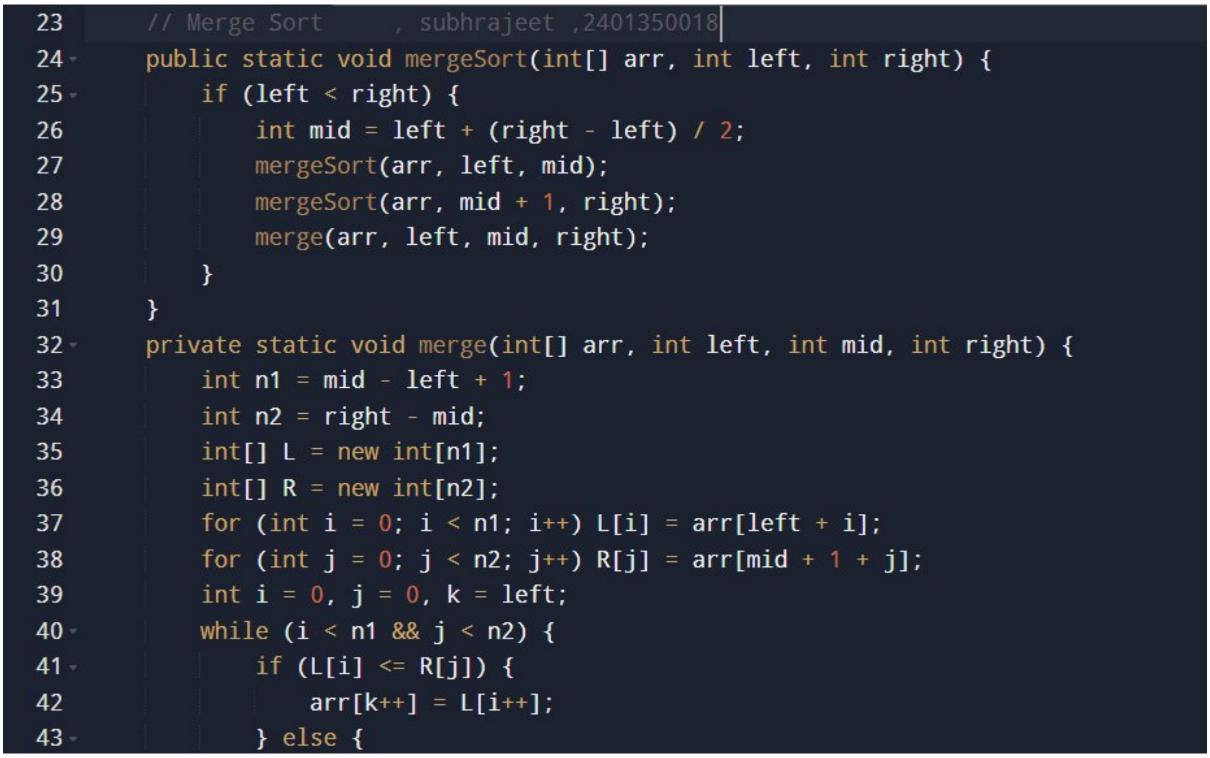


```
PS D:\Subhrajeet_2401350018> javac BasicSorts.java
PS D:\Subhrajeet_2401350018> java BasicSorts
Insertion Sort: [1, 2, 3, 4, 5, 6]
Selection Sort: [11, 12, 22, 25, 64]
Bubble Sort:    [1, 2, 4, 5, 8]
```

Q12. 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.

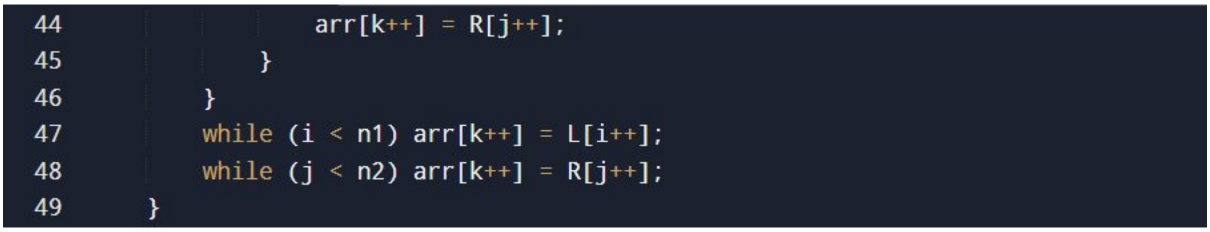


```java
public class AdvancedSorts {      // subhrajeet , 2401350018
    // Quick Sort
    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);  }}
    private 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++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;}
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;}
```

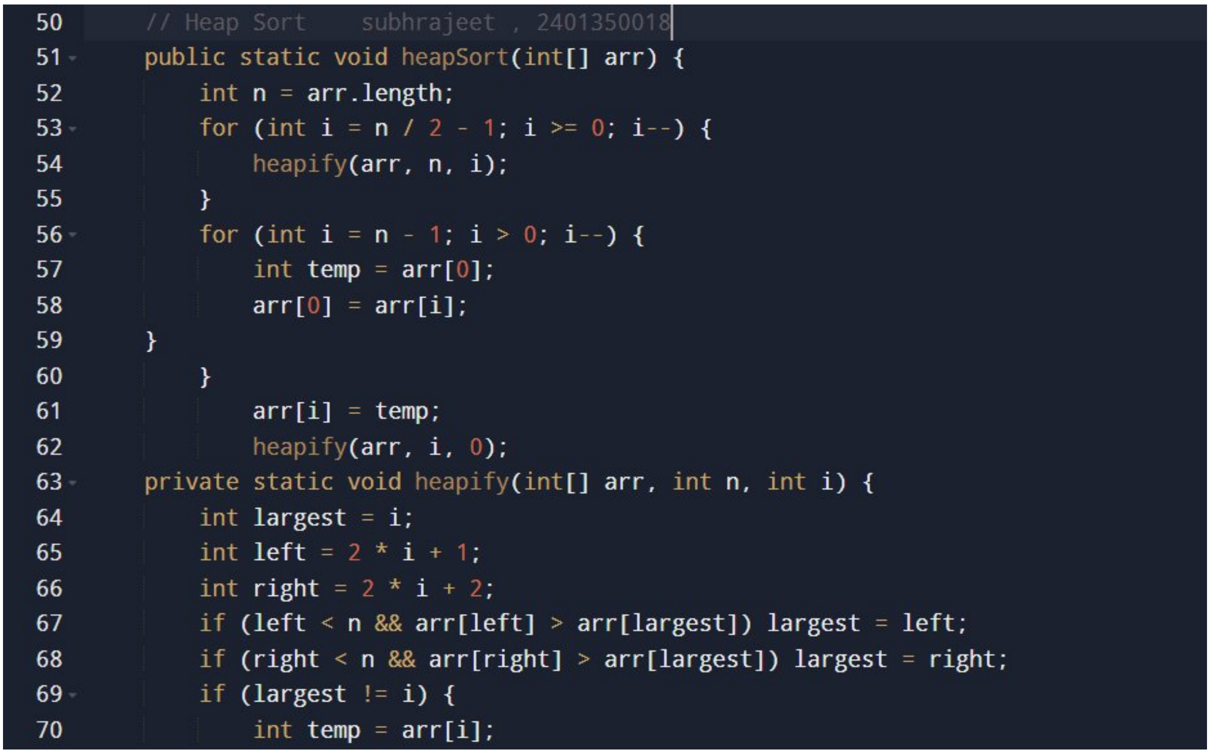


```java
    // Merge Sort      , subhrajeet ,2401350018
    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int mid = left + (right - left) / 2;
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);
            merge(arr, left, mid, right);
        }
    }
    private 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, k = left;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k++] = L[i++];
            } else {
```

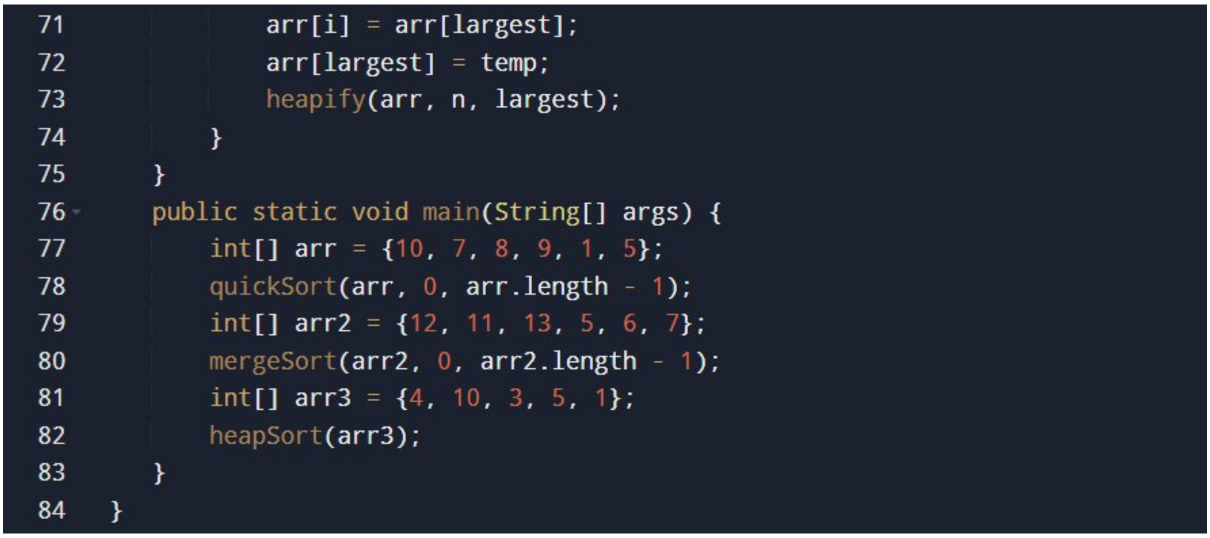
OUTPUT :



```
                arr[k++] = R[j++];
            }
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }
```

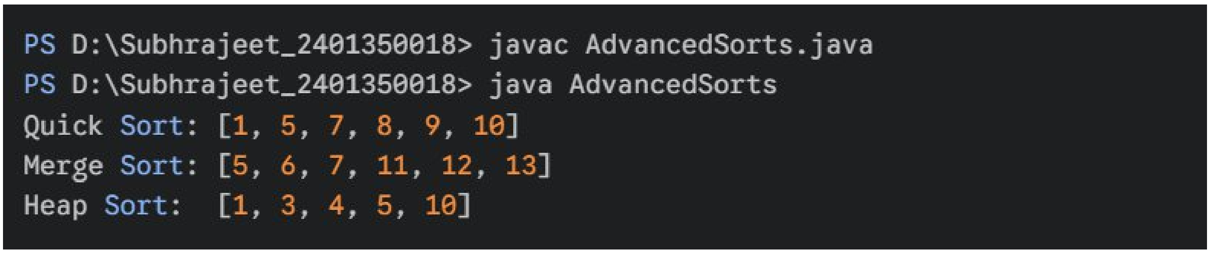


```
    // Heap Sort     subhrajeet , 2401350018
    public static void heapSort(int[] arr) {
        int n = arr.length;
        for (int i = n / 2 - 1; i >= 0; i--) {
            heapify(arr, n, i);
        }
        for (int i = n - 1; i > 0; i--) {
            int temp = arr[0];
            arr[0] = arr[i];
    }
        }
            arr[i] = temp;
            heapify(arr, i, 0);
    private static void heapify(int[] arr, int n, int i) {
        int largest = i;
        int left = 2 * i + 1;
        int right = 2 * i + 2;
        if (left < n && arr[left] > arr[largest]) largest = left;
        if (right < n && arr[right] > arr[largest]) largest = right;
        if (largest != i) {
            int temp = arr[i];
```



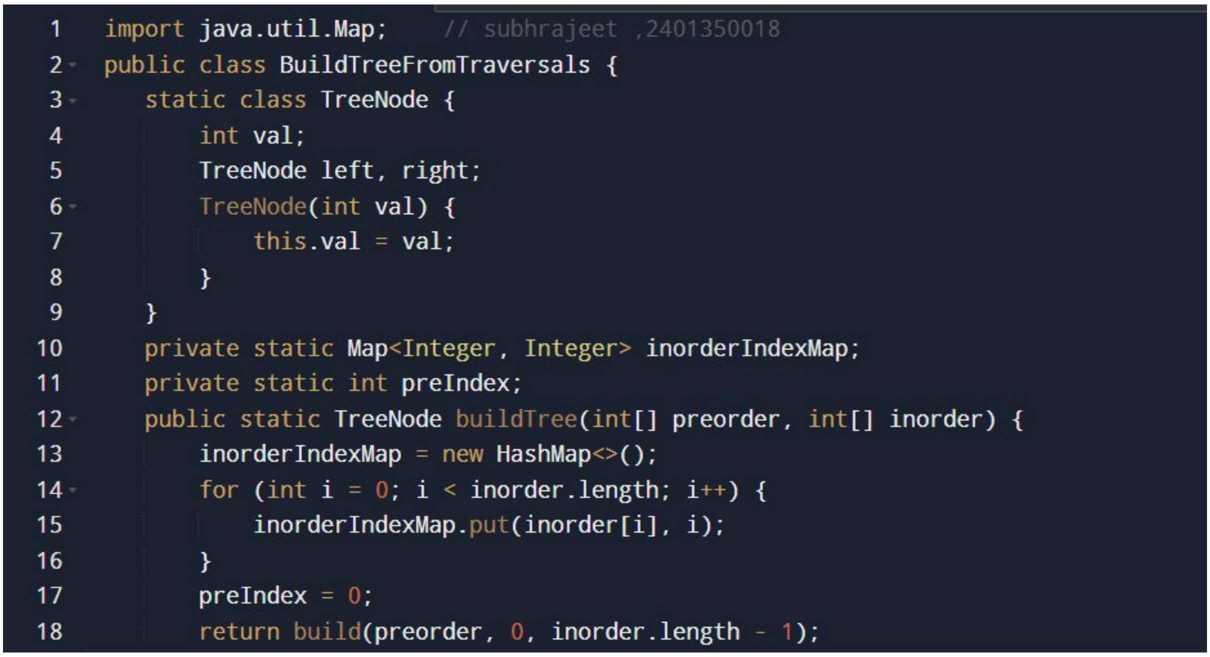
```
            arr[i] = arr[largest];
            arr[largest] = temp;
            heapify(arr, n, largest);
        }
    }
    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 9, 1, 5};
        quickSort(arr, 0, arr.length - 1);
        int[] arr2 = {12, 11, 13, 5, 6, 7};
        mergeSort(arr2, 0, arr2.length - 1);
        int[] arr3 = {4, 10, 3, 5, 1};
        heapSort(arr3);
    }
}
```

OUTPUT :

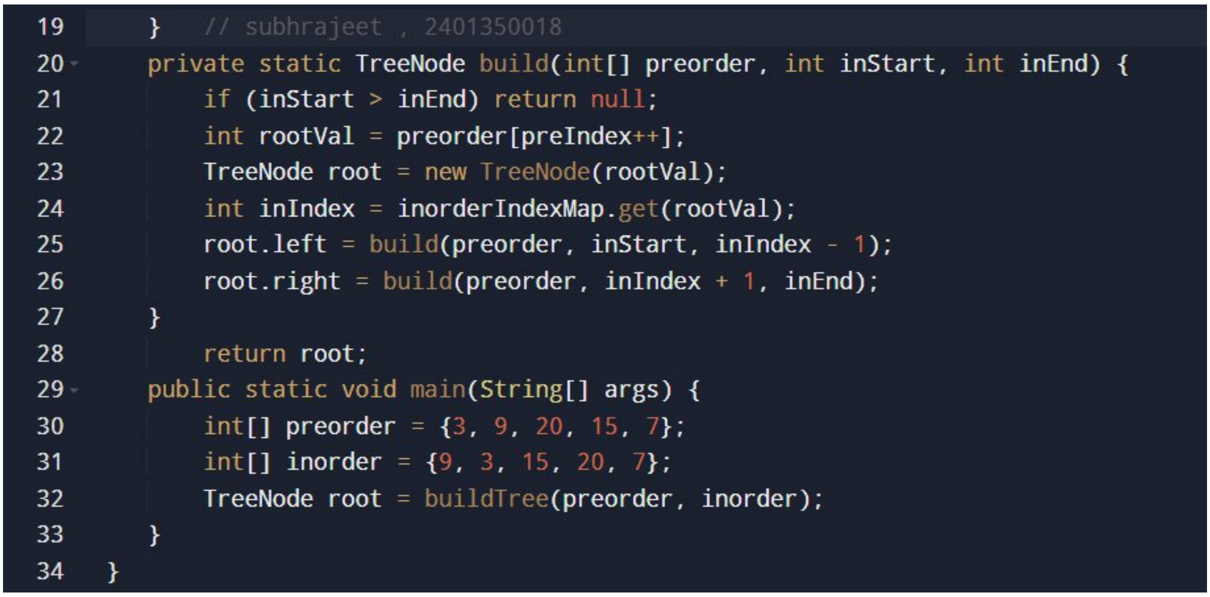


```
PS D:\Subhrajeet_2401350018> javac AdvancedSorts.java
PS D:\Subhrajeet_2401350018> java AdvancedSorts
Quick Sort: [1, 5, 7, 8, 9, 10]
Merge Sort: [5, 6, 7, 11, 12, 13]
Heap Sort:  [1, 3, 4, 5, 10]
```

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

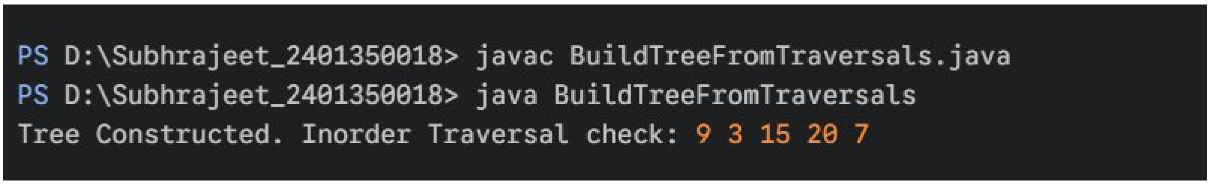


```java
import java.util.Map;     // subhrajeet ,2401350018
public class BuildTreeFromTraversals {
    static class TreeNode {
        int val;
        TreeNode left, right;
        TreeNode(int val) {
            this.val = val;
        }
    }
    private static Map<Integer, Integer> inorderIndexMap;
    private static int preIndex;
    public static TreeNode buildTree(int[] preorder, int[] inorder) {
        inorderIndexMap = new HashMap<>();
        for (int i = 0; i < inorder.length; i++) {
            inorderIndexMap.put(inorder[i], i);
        }
        preIndex = 0;
        return build(preorder, 0, inorder.length - 1);
```



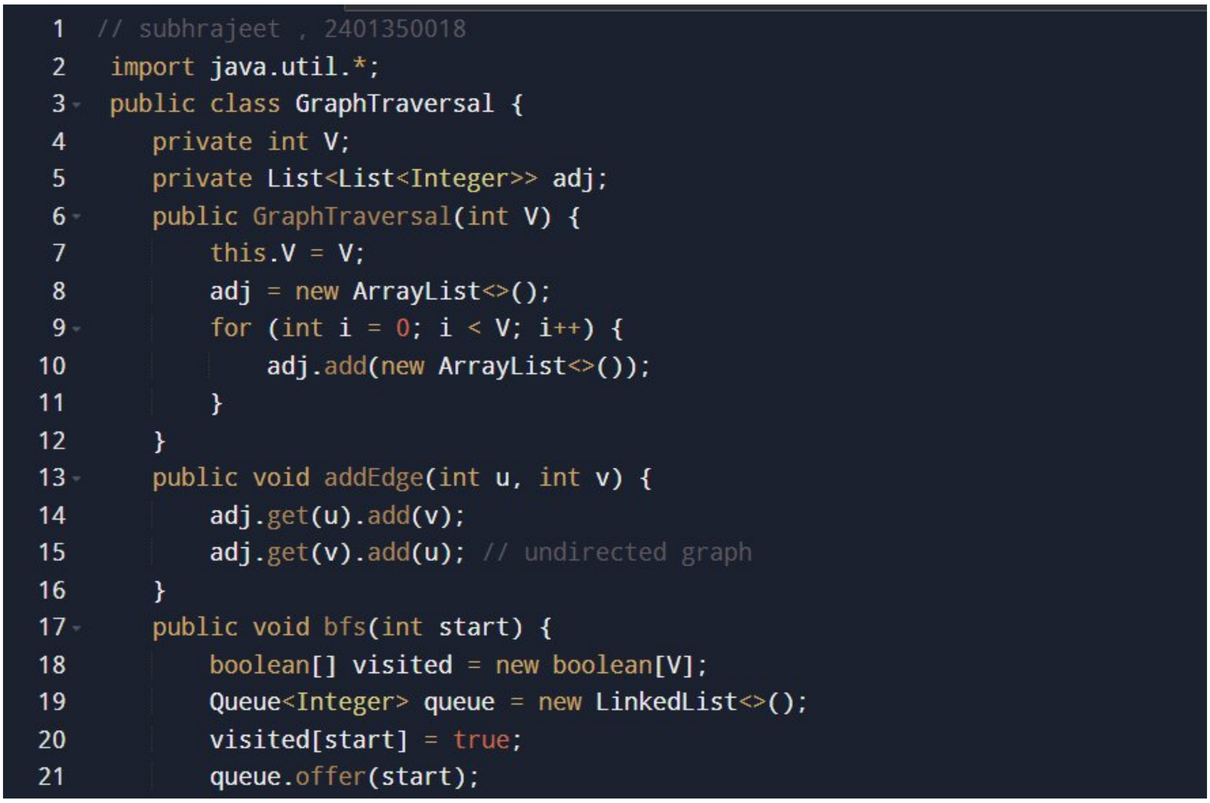
```java
    }   // subhrajeet , 2401350018
    private static TreeNode build(int[] preorder, int inStart, int inEnd) {
        if (inStart > inEnd) return null;
        int rootVal = preorder[preIndex++];
        TreeNode root = new TreeNode(rootVal);
        int inIndex = inorderIndexMap.get(rootVal);
        root.left = build(preorder, inStart, inIndex - 1);
        root.right = build(preorder, inIndex + 1, inEnd);
    }
        return root;
    public static void main(String[] args) {
        int[] preorder = {3, 9, 20, 15, 7};
        int[] inorder = {9, 3, 15, 20, 7};
        TreeNode root = buildTree(preorder, inorder);
    }
}
```

OUTPUT :

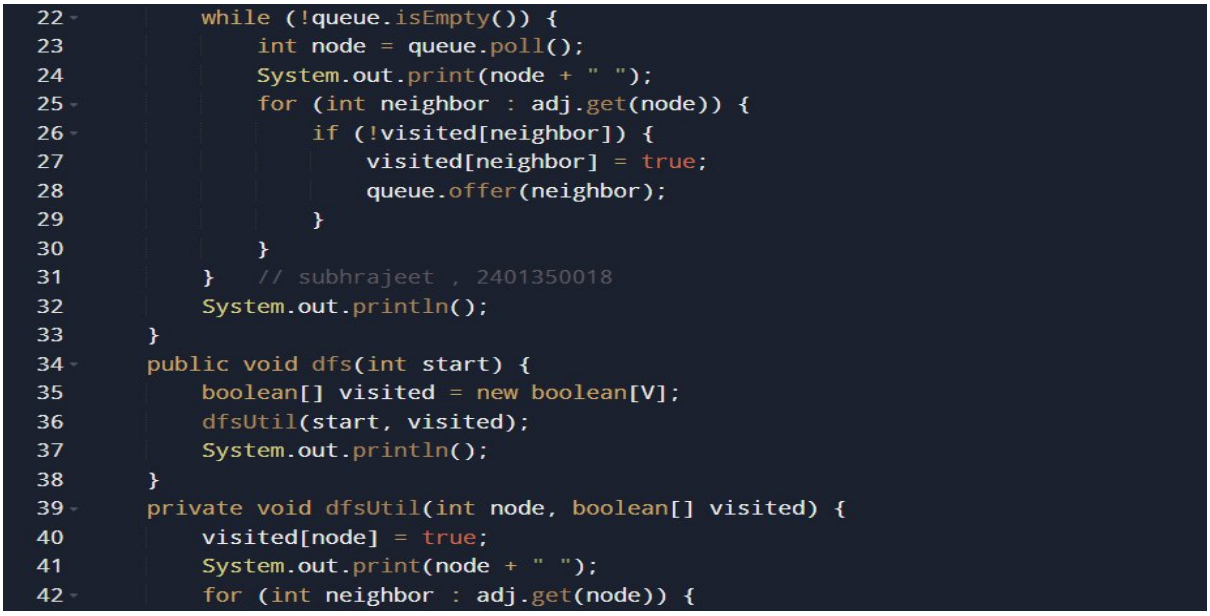


```
PS D:\Subhrajeet_2401350018> javac BuildTreeFromTraversals.java
PS D:\Subhrajeet_2401350018> java BuildTreeFromTraversals
Tree Constructed. Inorder Traversal check: 9 3 15 20 7
```

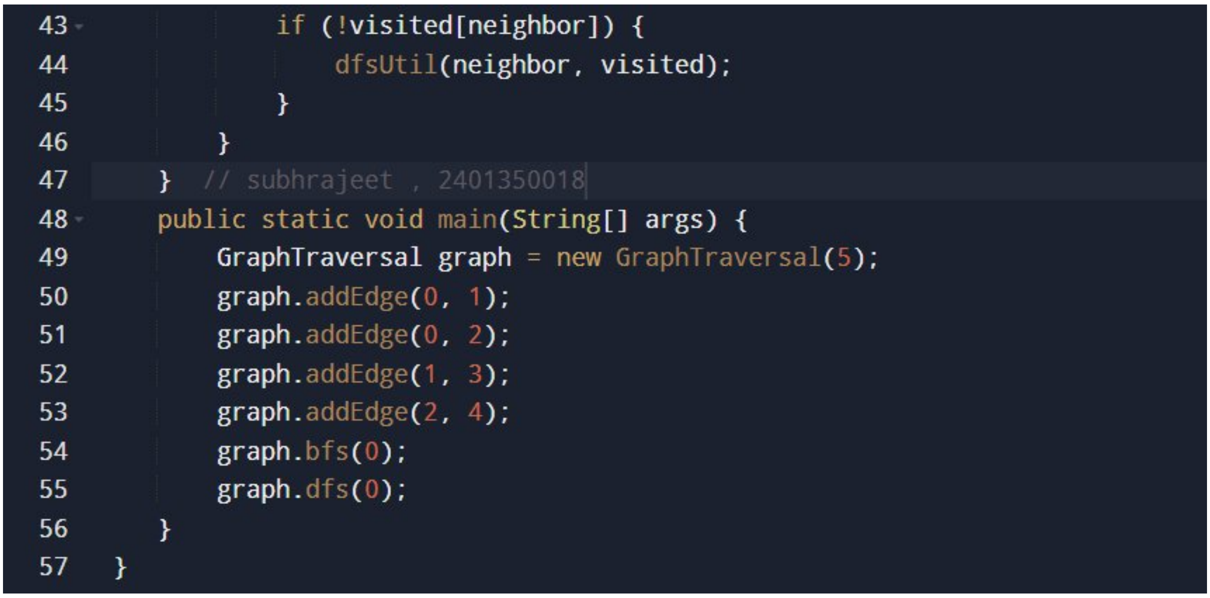
Q14 . Perform the traversal of graph(DFS,BFS)



```java
// subhrajeet , 2401350018
import java.util.*;
public class GraphTraversal {
    private int V;
    private List<List<Integer>> adj;
    public GraphTraversal(int V) {
        this.V = V;
        adj = new ArrayList<>();
        for (int i = 0; i < V; i++) {
            adj.add(new ArrayList<>());
        }
    }
    public void addEdge(int u, int v) {
        adj.get(u).add(v);
        adj.get(v).add(u); // undirected graph
    }
    public void bfs(int start) {
        boolean[] visited = new boolean[V];
        Queue<Integer> queue = new LinkedList<>();
        visited[start] = true;
        queue.offer(start);
```

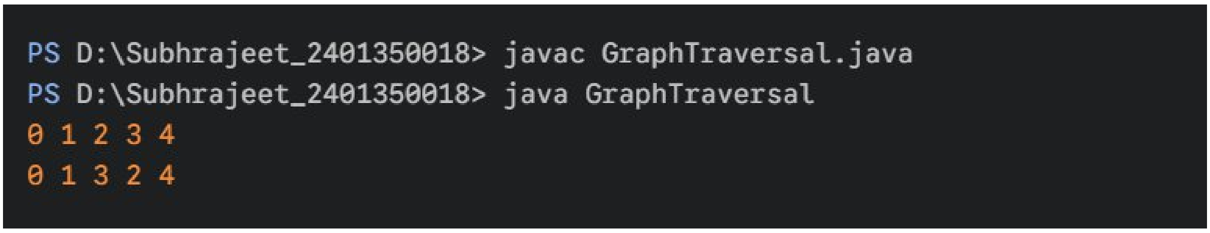


```java
        while (!queue.isEmpty()) {
            int node = queue.poll();
            System.out.print(node + " ");
            for (int neighbor : adj.get(node)) {
                if (!visited[neighbor]) {
                    visited[neighbor] = true;
                    queue.offer(neighbor);
                }
            }
        }   // subhrajeet , 2401350018
        System.out.println();
    }
    public void dfs(int start) {
        boolean[] visited = new boolean[V];
        dfsUtil(start, visited);
        System.out.println();
    }
    private void dfsUtil(int node, boolean[] visited) {
        visited[node] = true;
        System.out.print(node + " ");
        for (int neighbor : adj.get(node)) {
```



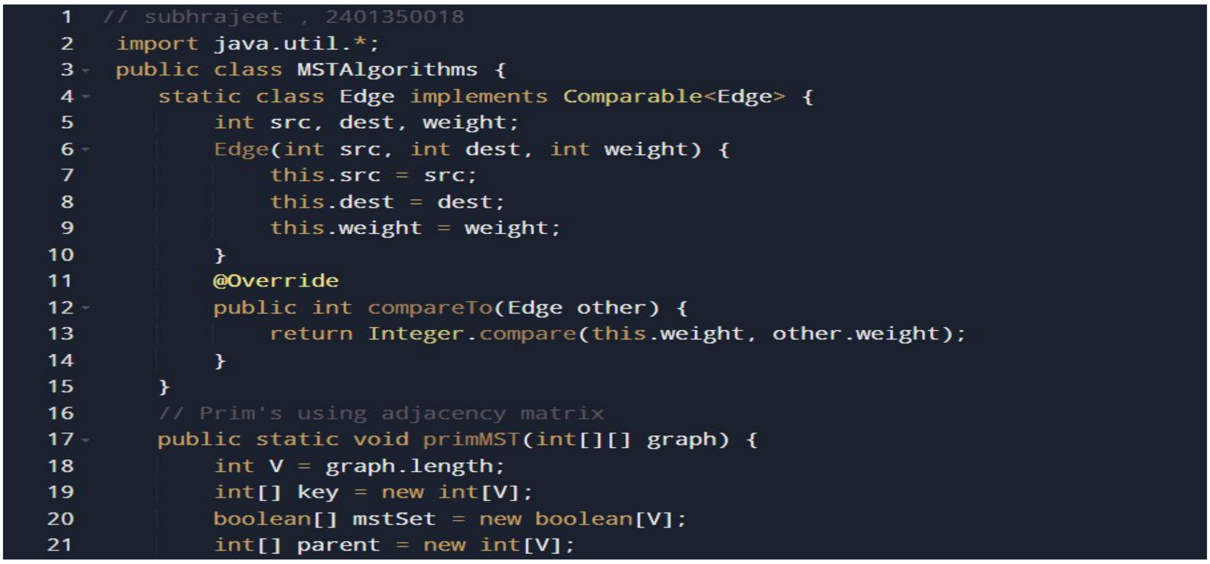
```java
            if (!visited[neighbor]) {
                dfsUtil(neighbor, visited);
            }
        }
    } // subhrajeet , 2401350018
    public static void main(String[] args) {
        GraphTraversal graph = new GraphTraversal(5);
        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(1, 3);
        graph.addEdge(2, 4);
        graph.bfs(0);
        graph.dfs(0);
    }
}
```

OUTPUT :

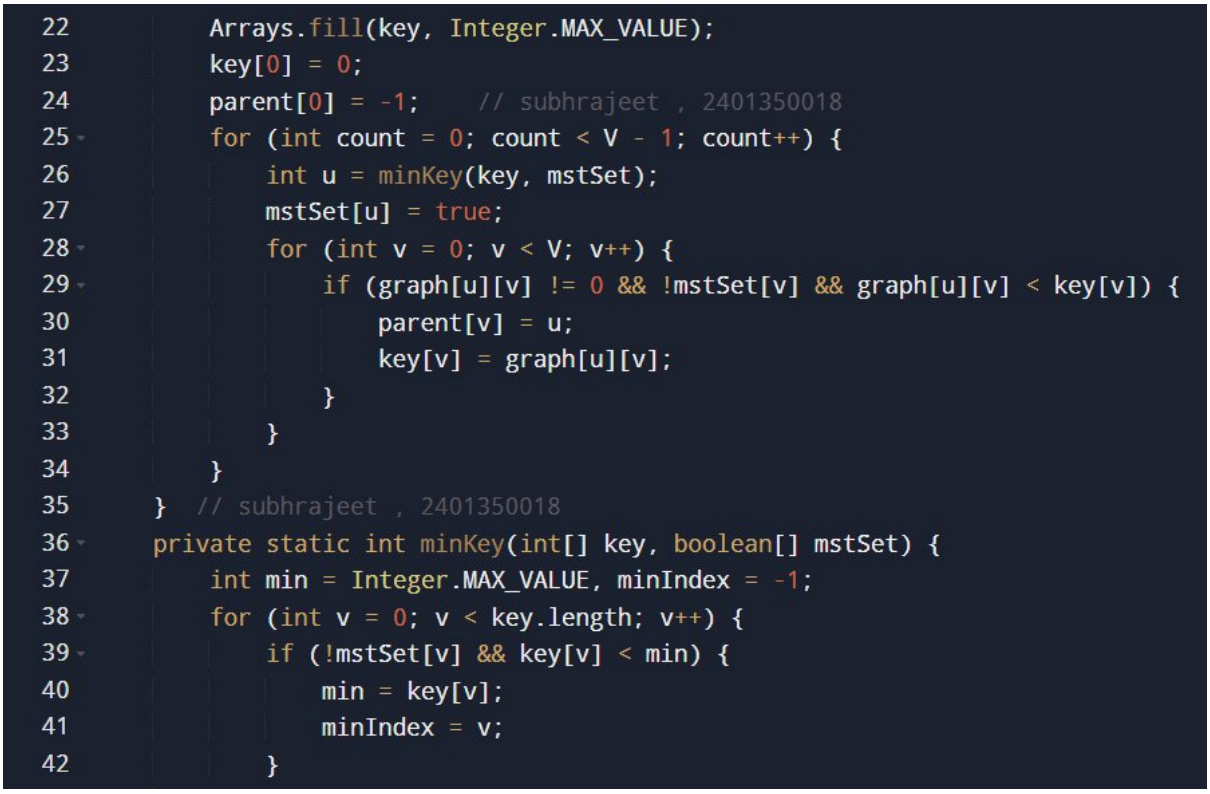


```
PS D:\Subhrajeet_2401350018> javac GraphTraversal.java
PS D:\Subhrajeet_2401350018> java GraphTraversal
0 1 2 3 4
0 1 3 2 4
```

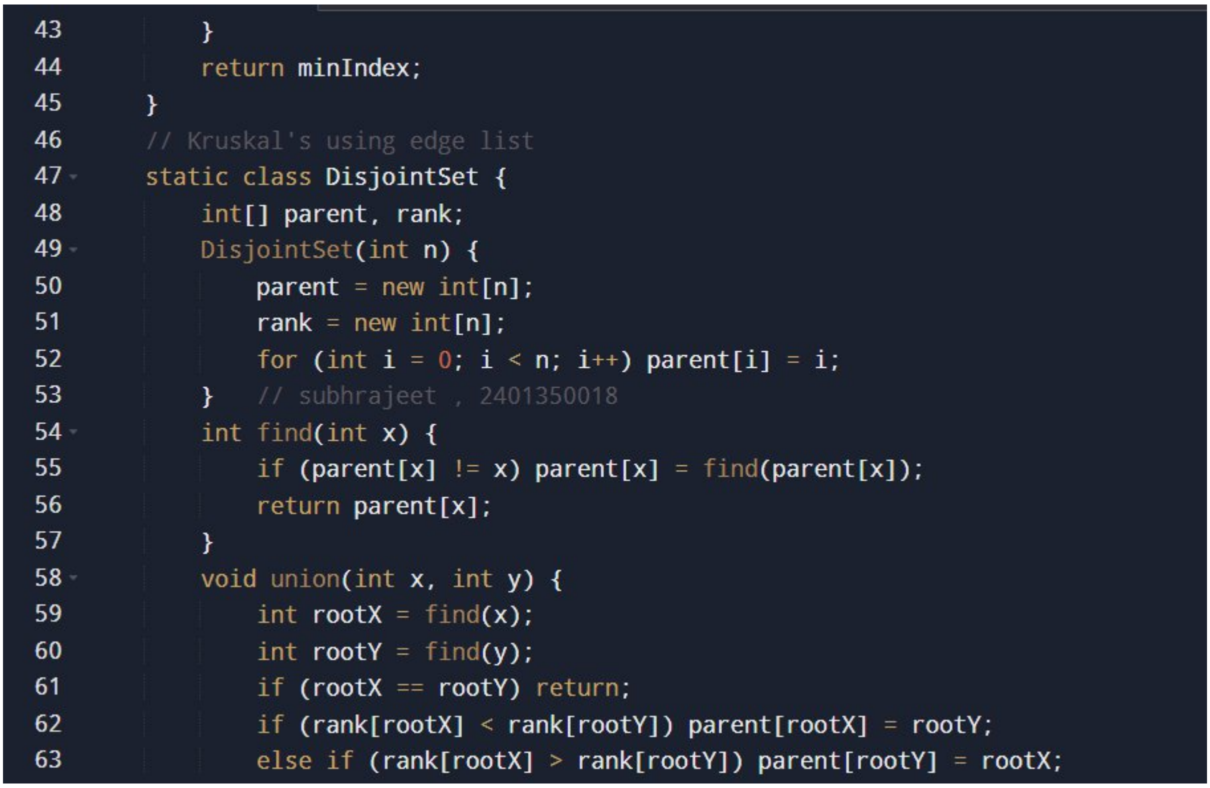
Q15 . Implement Prim's and Kruskal’s algorithm to find the minimum spanning tree of a graph.



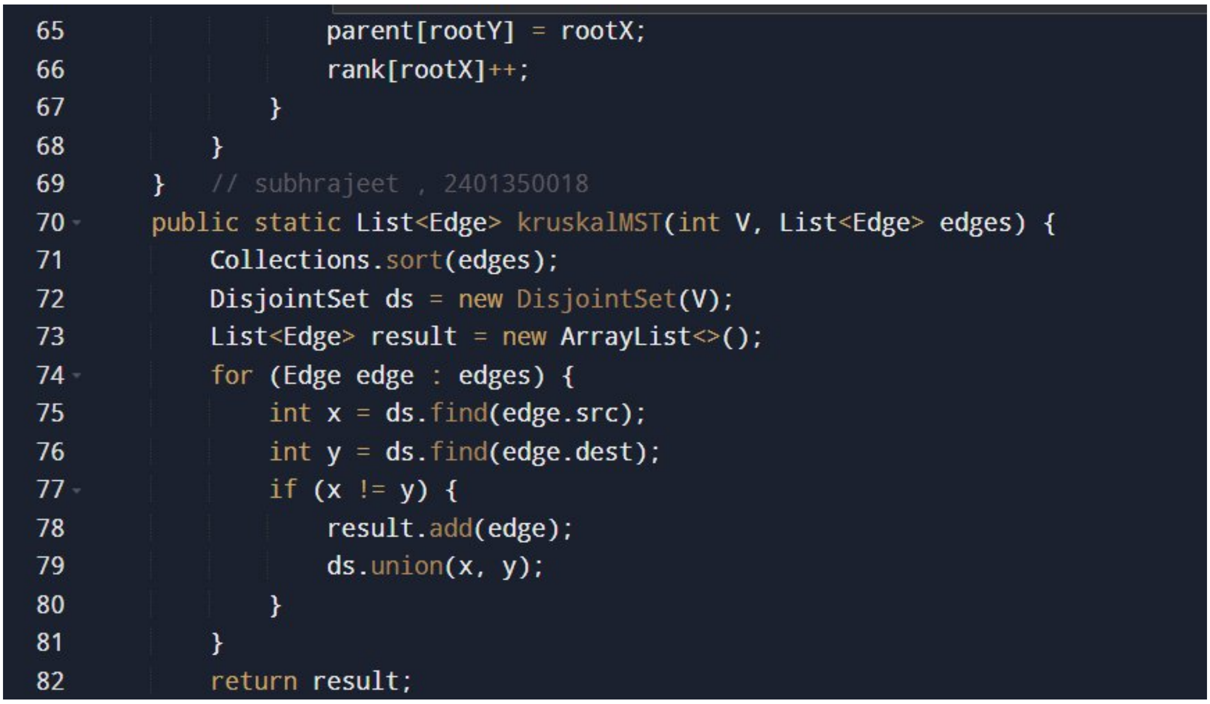
```java
// subhrajeet , 2401350018
import java.util.*;
public class MSTAlgorithms {
    static class Edge implements Comparable<Edge> {
        int src, dest, weight;
        Edge(int src, int dest, int weight) {
            this.src = src;
            this.dest = dest;
            this.weight = weight;
        }
        @Override
        public int compareTo(Edge other) {
            return Integer.compare(this.weight, other.weight);
        }
    }
    // Prim's using adjacency matrix
    public static void primMST(int[][] graph) {
        int V = graph.length;
        int[] key = new int[V];
        boolean[] mstSet = new boolean[V];
        int[] parent = new int[V];
```



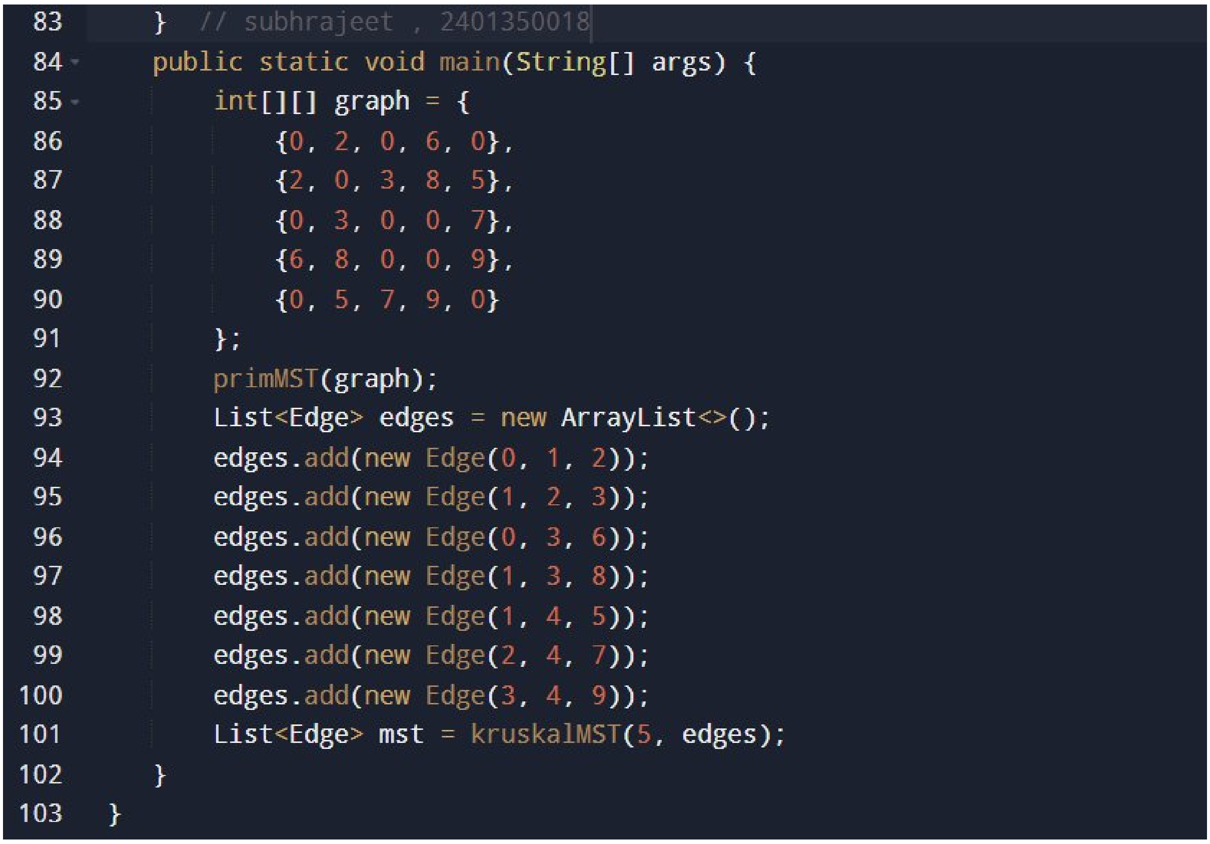
```java
        Arrays.fill(key, Integer.MAX_VALUE);
        key[0] = 0;
        parent[0] = -1;    // subhrajeet , 2401350018
        for (int count = 0; count < V - 1; count++) {
            int u = minKey(key, mstSet);
            mstSet[u] = true;
            for (int v = 0; v < V; v++) {
                if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
            }
        }
    }  // subhrajeet , 2401350018
    private static int minKey(int[] key, boolean[] mstSet) {
        int min = Integer.MAX_VALUE, minIndex = -1;
        for (int v = 0; v < key.length; v++) {
            if (!mstSet[v] && key[v] < min) {
                min = key[v];
                minIndex = v;
            }
```



```java
        }
        return minIndex;
    }
    // Kruskal's using edge list
    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;
        }   // subhrajeet , 2401350018
        int find(int x) {
            if (parent[x] != x) parent[x] = find(parent[x]);
            return parent[x];
        }
        void union(int x, int y) {
            int rootX = find(x);
            int rootY = find(y);
            if (rootX == rootY) return;
            if (rank[rootX] < rank[rootY]) parent[rootX] = rootY;
            else if (rank[rootX] > rank[rootY]) parent[rootY] = rootX;
```

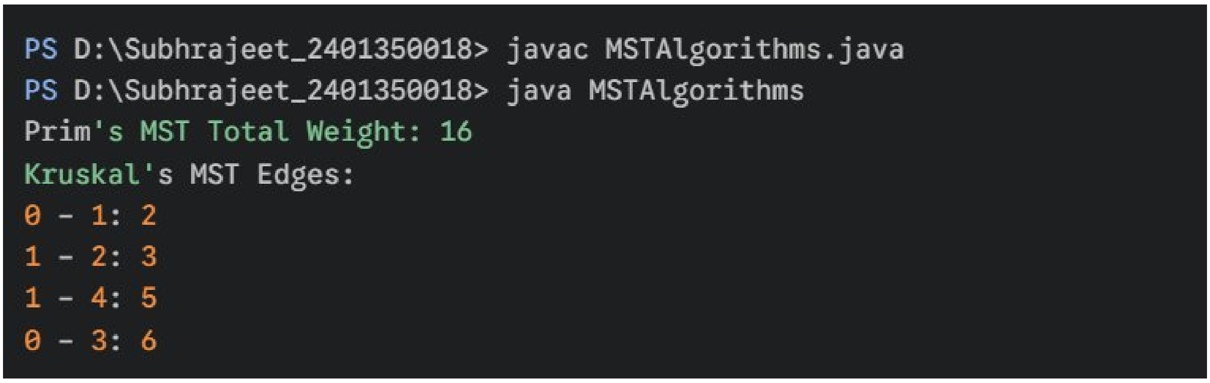


```
                parent[rootY] = rootX;
                rank[rootX]++;
            }
        }
    }   // subhrajeet , 2401350018
    public static List<Edge> kruskalMST(int V, List<Edge> edges) {
        Collections.sort(edges);
        DisjointSet ds = new DisjointSet(V);
        List<Edge> result = new ArrayList<>();
        for (Edge edge : edges) {
            int x = ds.find(edge.src);
            int y = ds.find(edge.dest);
            if (x != y) {
                result.add(edge);
                ds.union(x, y);
            }
        }
        return result;
```



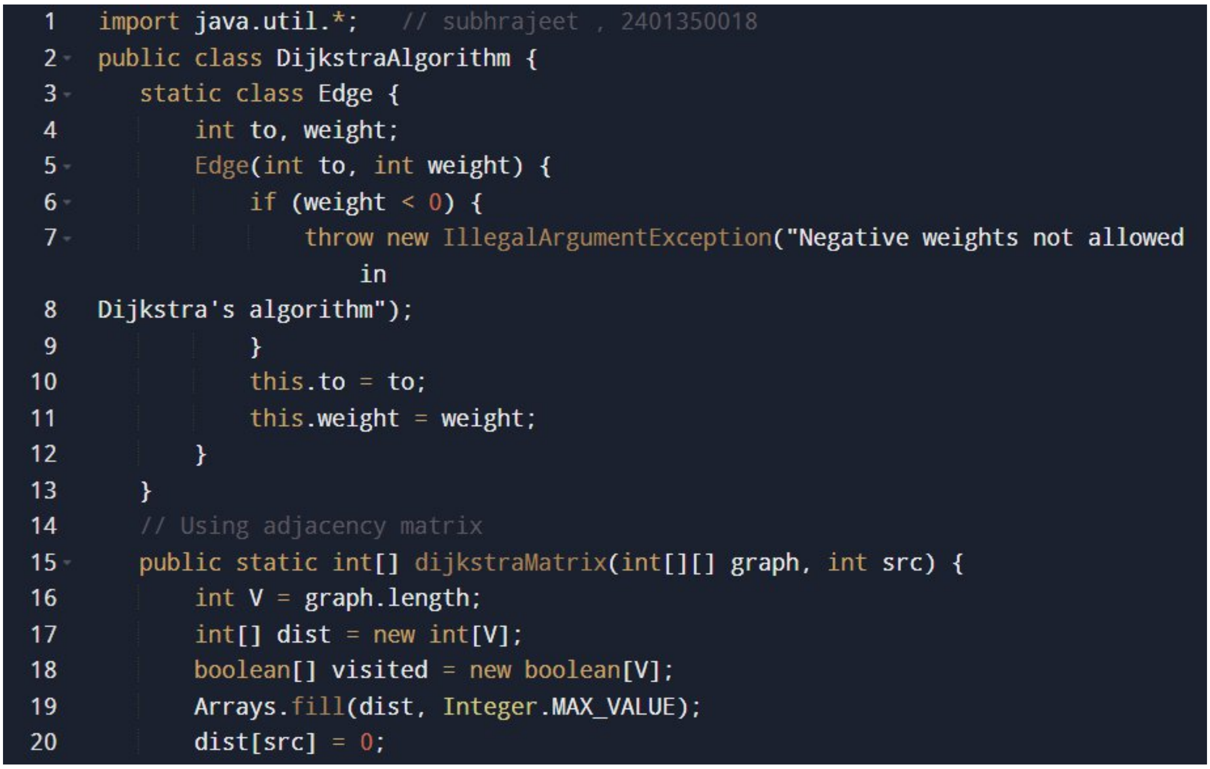
```
    }  // subhrajeet , 2401350018
    public static void main(String[] args) {
        int[][] graph = {
            {0, 2, 0, 6, 0},
            {2, 0, 3, 8, 5},
            {0, 3, 0, 0, 7},
            {6, 8, 0, 0, 9},
            {0, 5, 7, 9, 0}
        };
        primMST(graph);
        List<Edge> edges = new ArrayList<>();
        edges.add(new Edge(0, 1, 2));
        edges.add(new Edge(1, 2, 3));
        edges.add(new Edge(0, 3, 6));
        edges.add(new Edge(1, 3, 8));
        edges.add(new Edge(1, 4, 5));
        edges.add(new Edge(2, 4, 7));
        edges.add(new Edge(3, 4, 9));
        List<Edge> mst = kruskalMST(5, edges);
    }
}
```

OUTPUT :

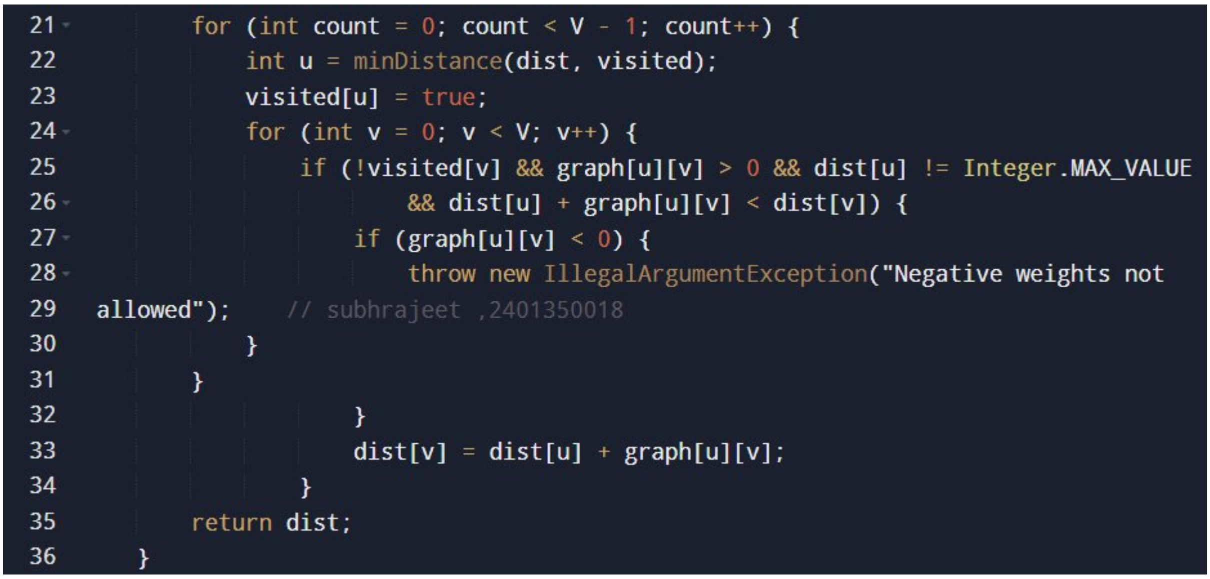


```
PS D:\Subhrajeet_2401350018> javac MSTAlgorithms.java
PS D:\Subhrajeet_2401350018> java MSTAlgorithms
Prim's MST Total Weight: 16
Kruskal's MST Edges:
0 - 1: 2
1 - 2: 3
1 - 4: 5
0 - 3: 6
```

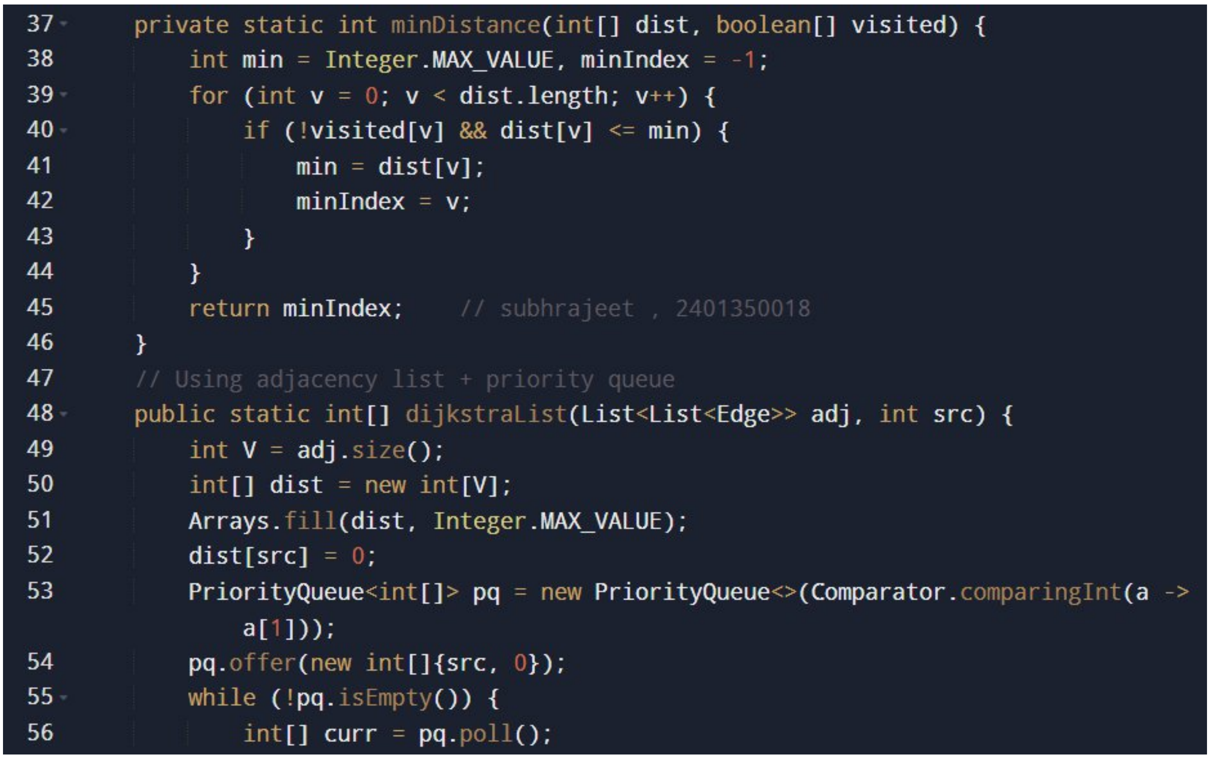
Q16 . Implement Dijkstra’s algorithm to find the shortest path from a source vertex to all other vertices in a weighted graph. Use both adjacency matrix and adjacency list representations for the graph. Ensure the algorithm handles negative weights appropriately.



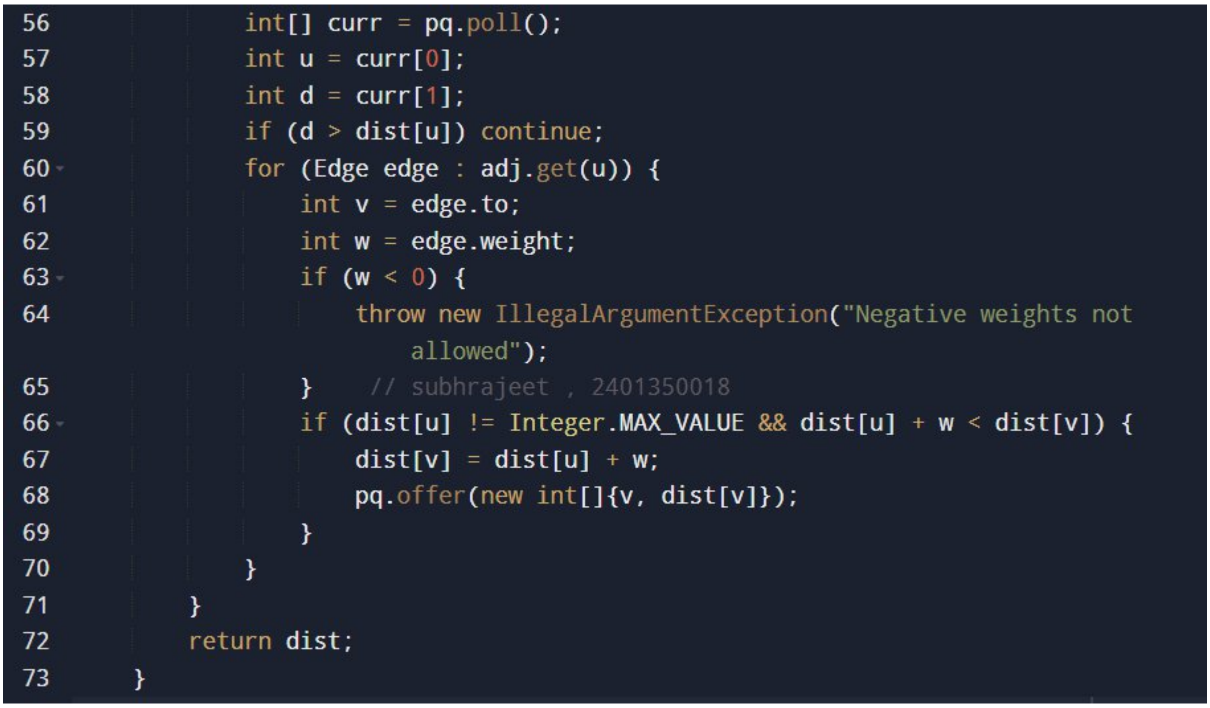
```
import java.util.*;   // subhrajeet , 2401350018
public class DijkstraAlgorithm {
    static class Edge {
        int to, weight;
        Edge(int to, int weight) {
            if (weight < 0) {
                throw new IllegalArgumentException("Negative weights not allowed
                    in
Dijkstra's algorithm");
            }
            this.to = to;
            this.weight = weight;
        }
    }
    // Using adjacency matrix
    public static int[] dijkstraMatrix(int[][] graph, int src) {
        int V = graph.length;
        int[] dist = new int[V];
        boolean[] visited = new boolean[V];
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[src] = 0;
```



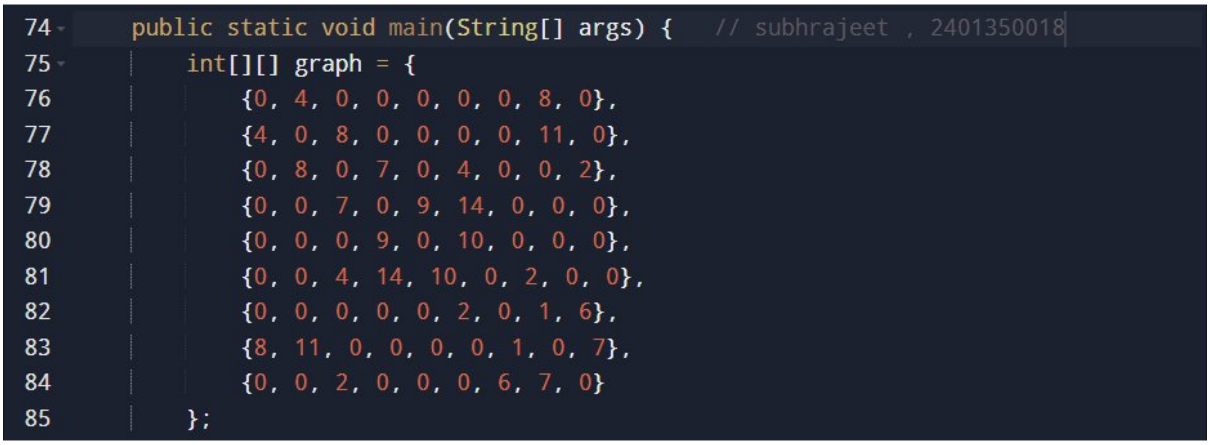
```
        for (int count = 0; count < V - 1; count++) {
            int u = minDistance(dist, visited);
            visited[u] = true;
            for (int v = 0; v < V; v++) {
                if (!visited[v] && graph[u][v] > 0 && dist[u] != Integer.MAX_VALUE
                        && dist[u] + graph[u][v] < dist[v]) {
                    if (graph[u][v] < 0) {
                        throw new IllegalArgumentException("Negative weights not
allowed");    // subhrajeet ,2401350018
            }
        }
                    }
                    dist[v] = dist[u] + graph[u][v];
                }
        return dist;
    }
```



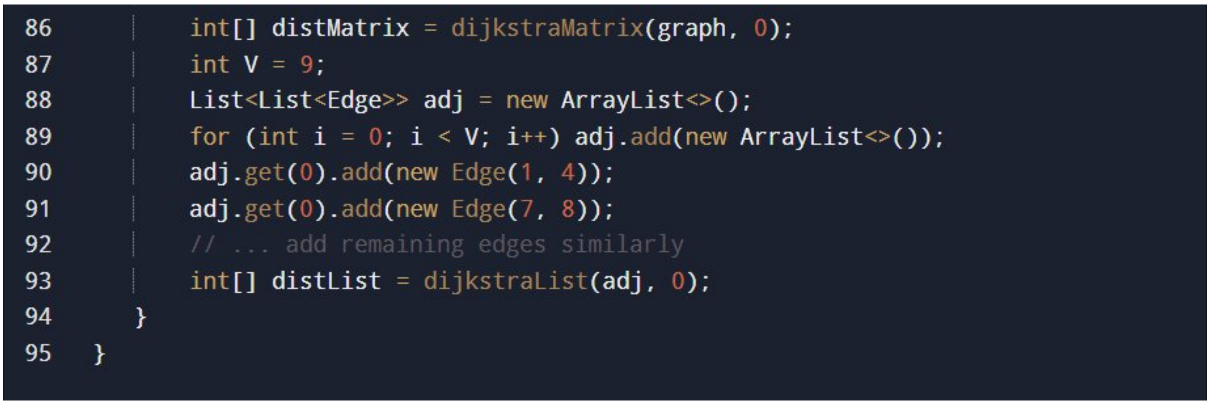
```
    private static int minDistance(int[] dist, boolean[] visited) {
        int min = Integer.MAX_VALUE, minIndex = -1;
        for (int v = 0; v < dist.length; v++) {
            if (!visited[v] && dist[v] <= min) {
                min = dist[v];
                minIndex = v;
            }
        }
        return minIndex;     // subhrajeet , 2401350018
    }
    // Using adjacency list + priority queue
    public static int[] dijkstraList(List<List<Edge>> adj, int src) {
        int V = adj.size();
        int[] dist = new int[V];
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[src] = 0;
        PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a ->
            a[1]));
        pq.offer(new int[]{src, 0});
        while (!pq.isEmpty()) {
            int[] curr = pq.poll();
```



```java
            int[] curr = pq.poll();
            int u = curr[0];
            int d = curr[1];
            if (d > dist[u]) continue;
            for (Edge edge : adj.get(u)) {
                int v = edge.to;
                int w = edge.weight;
                if (w < 0) {
                    throw new IllegalArgumentException("Negative weights not allowed");
                }   // subhrajeet , 2401350018
                if (dist[u] != Integer.MAX_VALUE && dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                    pq.offer(new int[]{v, dist[v]});
                }
            }
        }
        return dist;
    }
```

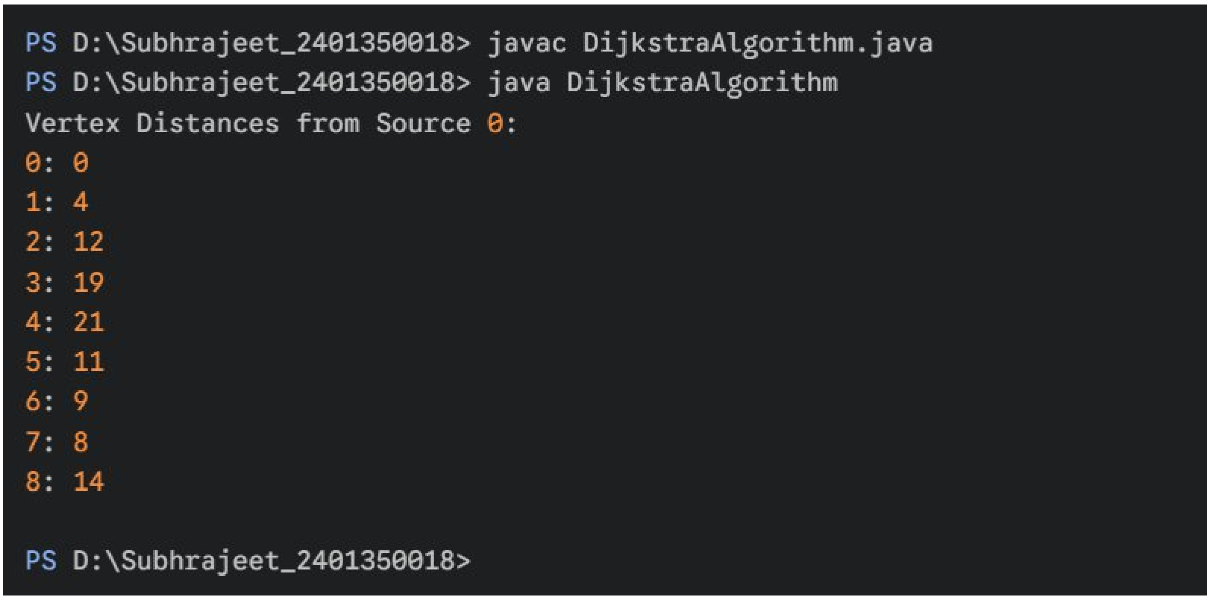


```java
    public static void main(String[] args) {   // subhrajeet , 2401350018
        int[][] graph = {
            {0, 4, 0, 0, 0, 0, 0, 8, 0},
            {4, 0, 8, 0, 0, 0, 0, 11, 0},
            {0, 8, 0, 7, 0, 4, 0, 0, 2},
            {0, 0, 7, 0, 9, 14, 0, 0, 0},
            {0, 0, 0, 9, 0, 10, 0, 0, 0},
            {0, 0, 4, 14, 10, 0, 2, 0, 0},
            {0, 0, 0, 0, 0, 2, 0, 1, 6},
            {8, 11, 0, 0, 0, 0, 1, 0, 7},
            {0, 0, 2, 0, 0, 0, 6, 7, 0}
        };
```



```java
        int[] distMatrix = dijkstraMatrix(graph, 0);
        int V = 9;
        List<List<Edge>> adj = new ArrayList<>();
        for (int i = 0; i < V; i++) adj.add(new ArrayList<>());
        adj.get(0).add(new Edge(1, 4));
        adj.get(0).add(new Edge(7, 8));
        // ... add remaining edges similarly
        int[] distList = dijkstraList(adj, 0);
    }
}
```

OUTPUT :



```
PS D:\Subhrajeet_2401350018> javac DijkstraAlgorithm.java
PS D:\Subhrajeet_2401350018> java DijkstraAlgorithm
Vertex Distances from Source 0:
0: 0
1: 4
2: 12
3: 19
4: 21
5: 11
6: 9
7: 8
8: 14

PS D:\Subhrajeet_2401350018>
```