Department of Computer Application

National Institute of Technology Kurukshetra, Kurukshetra -136119

Data Structures Lab

MCA-130

(2024-27)

Submitted by:

Name: *Shrushranto Rajbongshi*

Roll no: *524110026*

Semester: *2nd*

Section: *Group 2*

Submitted to:

Name: *Ms.Pooja*

Date: *29-04-2025*

Self-Declaration

I, **Shrushranto Rajbongshi**, bearing **Roll No. 524110026**, hereby declare that the **Data Structures Practical Programs** submitted by me are the result of my own sincere efforts, dedication, and understanding.

Each program included in this collection has been carefully designed, structured, and implemented to ensure **clarity**, **originality**, and **efficiency**. I have personally written every piece of code in the **Java programming language**, following proper coding standards, logical thinking, and best practices.

All solutions have been developed independently, with sincere efforts to apply the concepts I have learned through study and practice. I have referred to resources such as the ***Core Java* book**, classroom notes, and authorized academic materials to strengthen my understanding. Wherever necessary, I have applied my own logic, creativity, and problem-solving skills to design unique and optimized solutions. No part of the work has been copied from unauthorized sources or peers.

This submission has been prepared strictly in accordance with the academic guidelines and ethical practices prescribed by the institution. I fully accept responsibility for the authenticity and originality of the work presented.

I am confident that this submission reflects my true learning, hard work, and understanding of **Data Structures and Algorithms**.

I look forward to its evaluation by the respected faculty.

Date: 29-04-2025
Signature: Shrushranto Rajbongshi

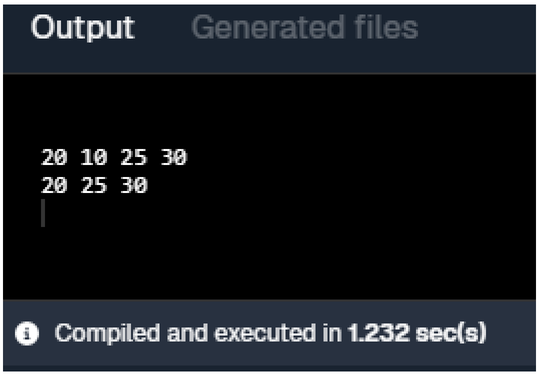
Contents

1. Write a program to perform insert, delete and traverse operations on the singly linked list in the beginning, end and on any specific location.

Code:-

```
public class SLL {
    class Node {
        int data;
        Node next;
        Node(int d) { data = d; }
    }
    Node head = null;
    void insertAtBegin(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
    }
    void insertAtEnd(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            return;
        }
        Node temp = head;
        while (temp.next != null) temp = temp.next;
        temp.next = newNode;
    }
    void insertAtPos(int data, int pos) {
        if (pos == 0) insertAtBegin(data);
        else {
            Node temp = head;
            for (int i = 0; i < pos - 1 && temp != null; i++) temp = temp.next;
            if (temp == null) return;
            Node newNode = new Node(data);
            newNode.next = temp.next;
            temp.next = newNode;
        }
    }
    void deleteAtPos(int pos) {
        if (head == null) return;
        if (pos == 0) {
            head = head.next;
            return;
        }
        Node temp = head;
        for (int i = 0; temp != null && i < pos - 1; i++) temp = temp.next;
        if (temp == null || temp.next == null) return;
        temp.next = temp.next.next;
    }
    void traverse() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        SLL list = new SLL();
        list.insertAtEnd(10);
        list.insertAtBegin(20);
        list.insertAtEnd(30);
        list.insertAtPos(25, 2);
        list.traverse();
        list.deleteAtPos(1);
        list.traverse();
    }
}
```

Output:-



```
20 10 25 30
20 25 30
```

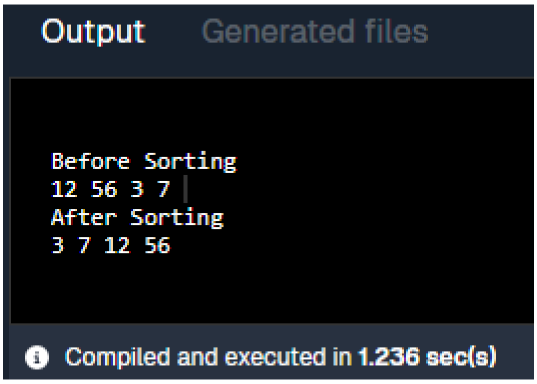
Compiled and executed in 1.232 sec(s)

2. Write a program to rearrange the elements of a singly linked list in ascending or descending order.

Code:-

```
import java.util.*;


public class RearrangeList {
    static class Node {
        int data;
        Node next;
        Node(int d) { data = d; }
    }


    static Node sortList(Node head, boolean ascending) {
        ArrayList<Integer> arr = new ArrayList<>();
        for (Node temp = head; temp != null; temp = temp.next)
arr.add(temp.data);
        if (ascending) Collections.sort(arr);
        else arr.sort(Collections.reverseOrder());
        Node temp = head;
        for (int val : arr) {
            temp.data = val;
            temp = temp.next;
        }
        return head;
    }
    static void printList(Node head) {
        for (Node temp = head; temp != null; temp = temp.next)
            System.out.print(temp.data + " ");
        System.out.println();
    }
    public static void main(String[] args) {
        Node head = new Node(12);
        head.next = new Node(56);
        head.next.next = new Node(3);
        head.next.next.next = new Node(7);
        System.out.println("Before Sorting");
        printList(head);
        head = sortList(head, true);
        System.out.println("After Sorting");
        printList(head);
    }
}
```

Output:-



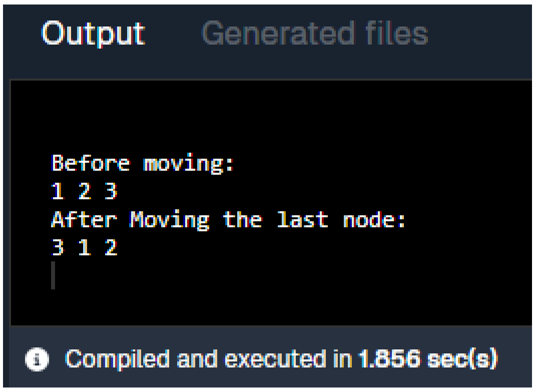
```
Before Sorting
12 56 3 7
After Sorting
3 7 12 56
```

3. Write a program to move the last node to the front of singly linked list.

Code:-

```
public class MoveLastToFront {
    static class Node {
        int data;
        Node next;
        Node(int d) { data = d; }
    }

    static Node moveLastToFront(Node head) {
        if (head == null | | head.next == null) return head;
        Node secLast = null, last = head;
        while (last.next != null) {
            secLast = last;
            last = last.next;
        }
        secLast.next = null;
        last.next = head;
        return last;
    }

    static void printList(Node head) {
        for (Node temp = head; temp != null; temp = temp.next)
            System.out.print(temp.data + " ");
        System.out.println();
    }

    public static void main(String[] args) {
        Node head = new Node(1);
        head.next = new Node(2);
        head.next.next = new Node(3);
        System.out.println("Before moving: ");
        printList(head);
        head = moveLastToFront(head);
        System.out.println("After Moving the last node: ");
        printList(head);
    }
}
```

Output:-



```
Before moving:
1 2 3
After Moving the last node:
3 1 2
```

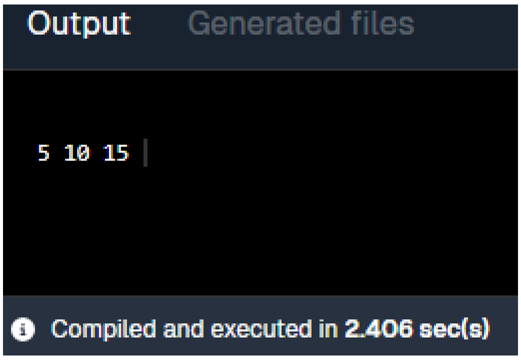
Compiled and executed in 1.856 sec(s)

4. Write a program to print the elements of singly link list using recursion.

Code:-

```
public class PrintListRec {
    static class Node {
        int data;
        Node next;
        Node(int d) { data = d; }
    }
    static void printRec(Node head) {
        if (head == null) return;
        System.out.print(head.data + " ");
        printRec(head.next);
    }
    public static void main(String[] args) {
        Node head = new Node(5);
        head.next = new Node(10);
        head.next.next = new Node(15);
        printRec(head);
    }
}
```

Output:-



```
5 10 15
```

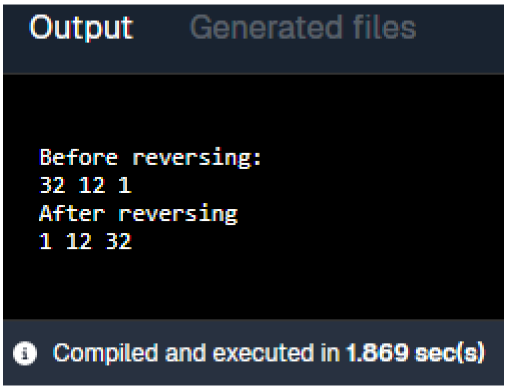
Compiled and executed in 2.406 sec(s)

5. Write a program to reverse link list using the iteration technique.

Code:-

```
public class ReverseIterative {
    static class Node {
        int data;
        Node next;
        Node(int d) { data = d; }
    }
    static Node reverse(Node head) {
        Node prev = null, curr = head;
        while (curr != null) {
            Node next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
    static void printList(Node head) {
        for (Node temp = head; temp != null; temp = temp.next)
            System.out.print(temp.data + " ");
        System.out.println();
    }
    public static void main(String[] args) {
        Node head = new Node(32);
        head.next = new Node(12);
        head.next.next = new Node(1);
        System.out.println("Before reversing:");
        printList(head);
        System.out.println("After reversing");
        head = reverse(head);
        printList(head);
    }
}
```

Output:-



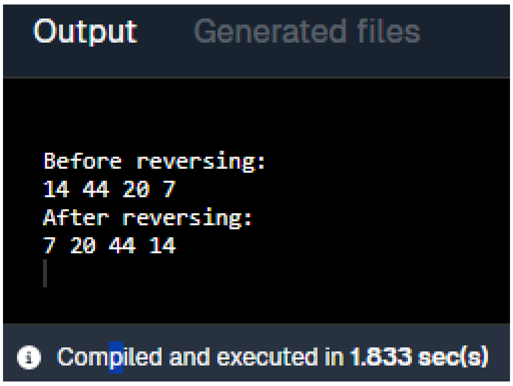
```
Before reversing:
32 12 1
After reversing
1 12 32
```

6. Write a program to reverse the singly link list using recursion.

Code:-

```
public class ReverseRecursively {
    static class Node {
        int data;
        Node next;
        Node(int d) { data = d; }
    }
    static Node reverse(Node head) {
        if (head == null || head.next == null) return head;
        Node newHead = reverse(head.next);
        head.next.next = head;
        head.next = null;
        return newHead;
    }
    static void printList(Node head) {
        for (Node temp = head; temp != null; temp = temp.next)
            System.out.print(temp.data + " ");
        System.out.println();
    }

    public static void main(String[] args) {
        Node head = new Node(14);
        head.next = new Node(44);
        head.next.next = new Node(20);
        head.next.next.next = new Node(7);
        System.out.println("Before reversing:");
        printList(head);
        System.out.println("After reversing:");
        head = reverse(head);
        printList(head);
    }
}
```

Output:-



```
Before reversing:
14 44 20 7
After reversing:
7 20 44 14
```

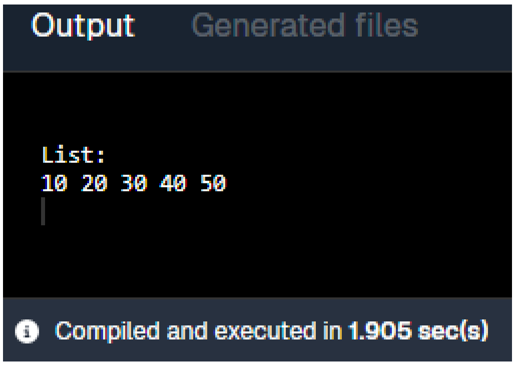
Compiled and executed in 1.833 sec(s)

7. Write a program to implement a circular linked list.

Code:-

```
public class CircularLinkedList {
    static class Node {
        int data;
        Node next;
        Node(int d) { data = d; }
    }

    Node last = null;

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

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

    public static void main(String[] args) {
        CircularLinkedList cll = new CircularLinkedList();
        cll.insert(10);
        cll.insert(20);
        cll.insert(30);
        cll.insert(40);
        cll.insert(50);
        System.out.println("List:");
        cll.display();
    }
}
```

Output:-



```
List:
10 20 30 40 50
```

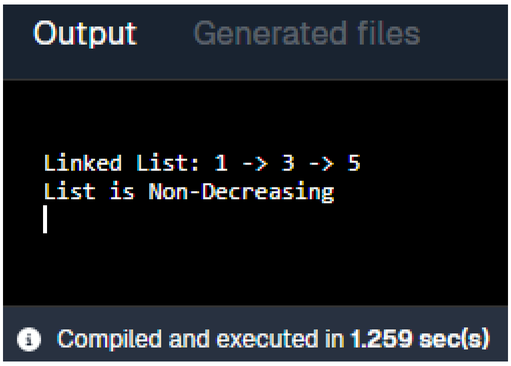
Compiled and executed in 1.905 sec(s)

8. Write a program to check whether the given singly linked list is in non-decreasing order or not.

Code:-

```
public class CheckNonDecreasing {
    static class Node {
        int data;
        Node next;
        Node(int d) { data = d; }
    }
    static boolean isNonDecreasing(Node head) {
        for (Node temp = head; temp != null && temp.next != null; temp = temp.next) {
            if (temp.data > temp.next.data) return false;
        }
        return true;
    }
    static void printList(Node head) {
        for (Node temp = head; temp != null; temp = temp.next) {
            System.out.print(temp.data);
            if (temp.next != null) System.out.print(" -> ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Node head = new Node(1);
        head.next = new Node(3);
        head.next.next = new Node(5);

        System.out.print("Linked List: ");
        printList(head);

        System.out.println(isNonDecreasing(head) ? "List is Non-Decreasing" :
"List is NOT Non-Decreasing");
    }
}
```

Output:-



```
Linked List: 1 -> 3 -> 5
List is Non-Decreasing
```

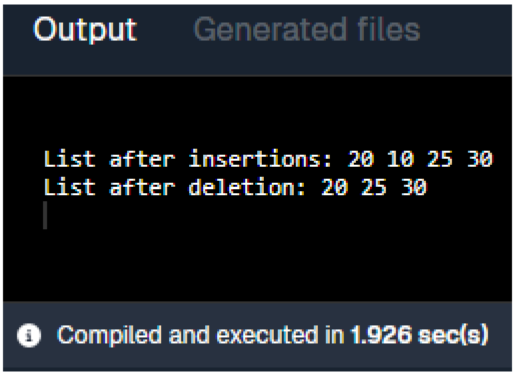
Compiled and executed in 1.259 sec(s)

9. Write a program to perform insert, delete, and traverse operations on the doubly linked list in the beginning, end and on any specific location.

Code:-

```
public class DLL {
    class Node {
        int data;
        Node next, prev;
        Node(int d) { data = d; }
    }

    Node head = null;

    void insertAtBegin(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        if (head != null) head.prev = newNode;
        head = newNode;
    }

    void insertAtEnd(int data) {
        Node newNode = new Node(data);
        if (head == null) { head = newNode; return; }
        Node temp = head;
        while (temp.next != null) temp = temp.next;
        temp.next = newNode;
        newNode.prev = temp;
    }
    void insertAtPos(int data, int pos) {
        if (pos == 0) insertAtBegin(data);
        else {
            Node temp = head;
            for (int i = 0; i < pos - 1 && temp != null; i++) temp = temp.next;
            if (temp == null) return;
            Node newNode = new Node(data);
            newNode.next = temp.next;
            if (temp.next != null) temp.next.prev = newNode;
            temp.next = newNode;
            newNode.prev = temp;
        }
    }
    void deleteAtPos(int pos) {
        if (head == null) return;
        Node temp = head;
        if (pos == 0) {
            head = temp.next;
            if (head != null) head.prev = null;
            return;
        }
        for (int i = 0; i < pos && temp != null; i++) temp = temp.next;
        if (temp == null) return;
        if (temp.next != null) temp.next.prev = temp.prev;
        if (temp.prev != null) temp.prev.next = temp.next;
    }

    void traverse() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        DLL list = new DLL();
        list.insertAtEnd(10);
        list.insertAtBegin(20);
        list.insertAtEnd(30);
        list.insertAtPos(25, 2);

        System.out.print("List after insertions: ");
        list.traverse();

        list.deleteAtPos(1);

        System.out.print("List after deletion: ");
        list.traverse();
    }
}
```

Output:-



```
List after insertions: 20 10 25 30
List after deletion: 20 25 30
```

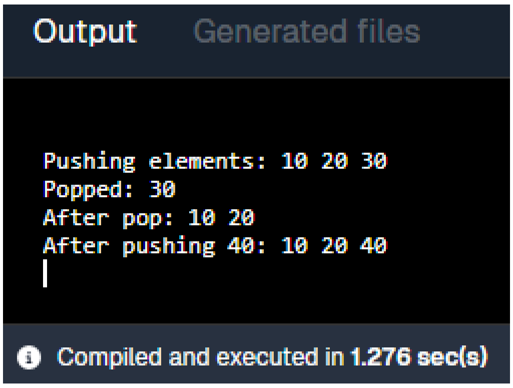
Compiled and executed in 1.926 sec(s)

10. Write a program to implement stack (push and pop operations) using array.

Code:-

```
public class StackArray {
    int top = -1;
    int[] stack = new int[5];

    void push(int data) {
        if (top == 4) {
            System.out.println("Stack Overflow");
            return;
        }
        stack[++top] = data;
    }
    void pop() {
        if (top == -1) {
            System.out.println("Stack Underflow");
            return;
        }
        System.out.println("Popped: " + stack[top--]);
    }

    void display() {
        if (top == -1) {
            System.out.println("Stack is empty");
            return;
        }
        for (int i = 0; i <= top; i++) {
            System.out.print(stack[i] + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        StackArray s = new StackArray();
        System.out.print("Pushing elements: ");
        s.push(10);
        s.push(20);
        s.push(30);
        s.display();
        s.pop();
        System.out.print("After pop: ");
        s.display();
        s.push(40);
        System.out.print("After pushing 40: ");
        s.display();
    }
}
```

Output:-



```
Pushing elements: 10 20 30
Popped: 30
After pop: 10 20
After pushing 40: 10 20 40
```

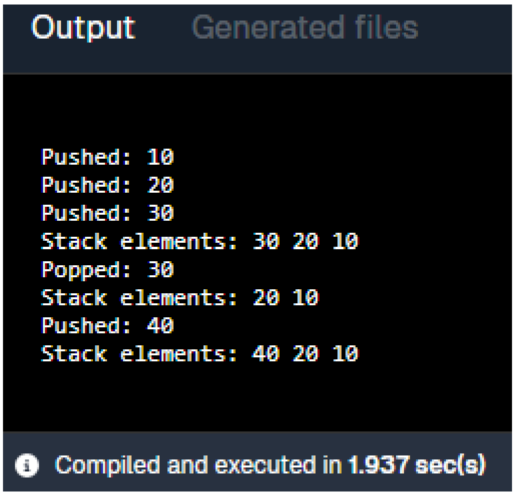
Compiled and executed in 1.276 sec(s)

11. Write a program to implement stack using singly linked list.

Code:-

```
public class StackLinkedList {
    class Node {
        int data;
        Node next;
        Node(int d) { data = d; }
    }

    Node top = null;
    void push(int data) {
        Node newNode = new Node(data);
        newNode.next = top;
        top = newNode;
        System.out.println("Pushed: " + data);
    }
    void pop() {
        if (top == null) {
            System.out.println("Stack Underflow");
            return;
        }
        System.out.println("Popped: " + top.data);
        top = top.next;
    }
    void display() {
        if (top == null) {
            System.out.println("Stack is empty");
            return;
        }
        Node temp = top;
        System.out.print("Stack elements: ");
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        StackLinkedList s = new StackLinkedList();

        s.push(10);
        s.push(20);
        s.push(30);
        s.display();
        s.pop();
        s.display();
        s.push(40);
        s.display();
    }
}
```

Output:-



```
Pushed: 10
Pushed: 20
Pushed: 30
Stack elements: 30 20 10
Popped: 30
Stack elements: 20 10
Pushed: 40
Stack elements: 40 20 10
```

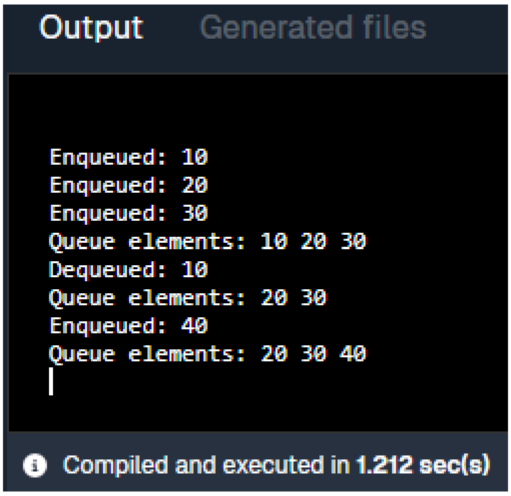
Compiled and executed in 1.937 sec(s)

12. Write a program to implement a queue using a circular array.

Code:-

```
public class CircularQueue {
    int[] q = new int[5];
    int front = -1, rear = -1, size = 5;

    void enqueue(int data) {
        if ((rear + 1) % size == front) {
            System.out.println("Queue Full");
            return;
        }
        if (front == -1) front = 0;
        rear = (rear + 1) % size;
        q[rear] = data;
        System.out.println("Enqueued: " + data);
    }

    void dequeue() {
        if (front == -1) {
            System.out.println("Queue Empty");
            return;
        }
        System.out.println("Dequeued: " + q[front]);
        if (front == rear) front = rear = -1;
        else front = (front + 1) % size;
    }

    void display() {
        if (front == -1) {
            System.out.println("Queue Empty");
            return;
        }
        int i = front;
        System.out.print("Queue elements: ");
        while (true) {
            System.out.print(q[i] + " ");
            if (i == rear) break;
            i = (i + 1) % size;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        CircularQueue cq = new CircularQueue();
        cq.enqueue(10);
        cq.enqueue(20);
        cq.enqueue(30);
        cq.display();
        cq.dequeue();
        cq.display();
        cq.enqueue(40);
        cq.display();
    }
}
```

Output:-



```
Enqueued: 10
Enqueued: 20
Enqueued: 30
Queue elements: 10 20 30
Dequeued: 10
Queue elements: 20 30
Enqueued: 40
Queue elements: 20 30 40
```

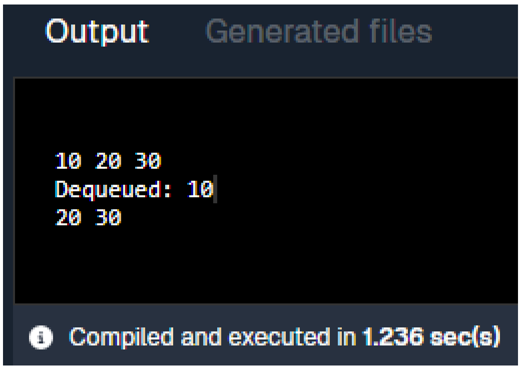
Compiled and executed in 1.212 sec(s)

13. Write a program to implement a queue using a circular linked list.

Code:-

```
public class CircularQueueLinkedList {
    class Node {
        int data;
        Node next;
        Node(int d) { data = d; }
    }

    Node rear = null;

    void enqueue(int data) {
        Node newNode = new Node(data);
        if (rear == null) {
            rear = newNode;
            rear.next = rear;
        } else {
            newNode.next = rear.next;
            rear.next = newNode;
            rear = newNode;
        }
    }

    void dequeue() {
        if (rear == null) {
            System.out.println("Queue Empty");
            return;
        }
        Node front = rear.next;
        if (rear == front) rear = null;
        else rear.next = front.next;
        System.out.println("Dequeued: " + front.data);
    }
    void display() {
        if (rear == null) {
            System.out.println("Queue Empty");
            return;
        }
        Node temp = rear.next;
        do {
            System.out.print(temp.data + " ");
            temp = temp.next;
        } while (temp != rear.next);
        System.out.println();
    }
    public static void main(String[] args) {
        CircularQueueLinkedList q = new CircularQueueLinkedList();
        q.enqueue(10);
        q.enqueue(20);
        q.enqueue(30);
        q.display();
        q.dequeue();
        q.display();
    }
}
```

Output:-



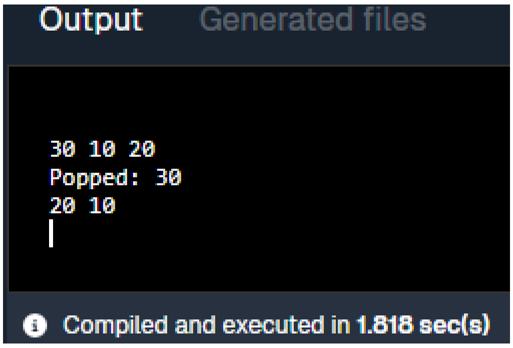
```
10 20 30
Dequeued: 10
20 30
```

14. Write a program to implement stack using priority queue.

Code:-

```
import java.util.PriorityQueue;
public class StackPriorityQueue {
    static class Element implements Comparable<Element> {
        int value, order;
        Element(int v, int o) { value = v; order = o; }

        public int compareTo(Element e) {
            return e.order - this.order;
        }
    }
    PriorityQueue<Element> pq = new PriorityQueue<>();
    int order = 0;
    void push(int data) {
        pq.add(new Element(data, ++order));
    }
    void pop() {
        if (pq.isEmpty()) {
            System.out.println("Stack Underflow");
            return;
        }
        System.out.println("Popped: " + pq.poll().value);
    }

    void display() {
        for (Element e : pq)
            System.out.print(e.value + " ");
        System.out.println();
    }
    public static void main(String[] args) {
        StackPriorityQueue s = new StackPriorityQueue();
        s.push(10);
        s.push(20);
        s.push(30);
        s.display();
        s.pop();
        s.display();
    }
}
```

Output:-



```
30 10 20
Popped: 30
20 10
```

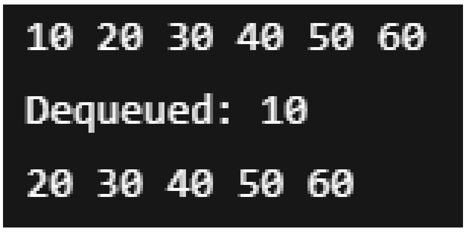
Compiled and executed in 1.818 sec(s)

15. Write a program to implement a queue using two stacks.

Code:-

```
import java.util.Stack;
public class QueueUsingStacks {
    Stack<Integer> s1 = new Stack<>();
    Stack<Integer> s2 = new Stack<>();
    void enqueue(int data) {
        s1.push(data);
    }
    void dequeue() {
        if (s1.isEmpty() && s2.isEmpty()) {
            System.out.println("Queue Empty");
            return;
        }
        if (s2.isEmpty()) {
            while (!s1.isEmpty()) s2.push(s1.pop());
        }
        System.out.println("Dequeued: " + s2.pop());
    }
    void display() {
        if (!s2.isEmpty()) {
            Stack<Integer> temp = (Stack<Integer>) s2.clone();
            while (!temp.isEmpty()) System.out.print(temp.pop() + " ");
        }
        Stack<Integer> temp = (Stack<Integer>) s1.clone();
        Stack<Integer> temp2 = new Stack<>();
        while (!temp.isEmpty()) temp2.push(temp.pop());
        while (!temp2.isEmpty()) System.out.print(temp2.pop() + " ");
        System.out.println();
    }
    public static void main(String[] args) {
        QueueUsingStacks q = new QueueUsingStacks();
        q.enqueue(10);
        q.enqueue(20);
        q.enqueue(30);
        q.enqueue(40);
        q.enqueue(50);
        q.enqueue(60);
        q.display();
        q.dequeue();
        q.display();
    }
}
```

Output:-



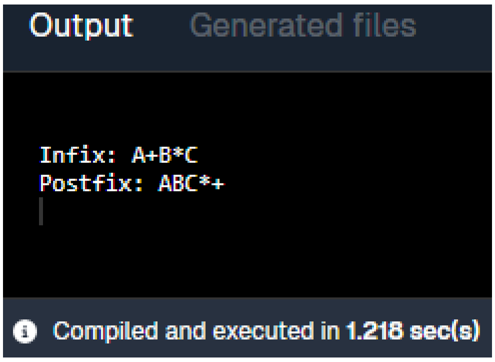
```
10 20 30 40 50 60
Dequeued: 10
20 30 40 50 60
```

16. Write a program to convert an infix expression to a postfix expression.

Code:-

```
import java.util.Stack;
public class InfixToPostfix {
    static int precedence(char ch) {
        if (ch == '+' || ch == '-') return 1;
        if (ch == '*' || ch == '/') return 2;
        return 0;
    }
    static String convert(String exp) {
        Stack<Character> stack = new Stack<>();
        StringBuilder result = new StringBuilder();
        for (char c : exp.toCharArray()) {
            if (Character.isLetterOrDigit(c)) result.append(c);
            else if (c == '(') stack.push(c);
            else if (c == ')') {
                while (stack.peek() != '(') result.append(stack.pop());
                stack.pop();
            } else {
                while (!stack.isEmpty() && precedence(c) <=
precedence(stack.peek()))
                    result.append(stack.pop());
                stack.push(c);
            }
        }
        while (!stack.isEmpty()) result.append(stack.pop());
        return result.toString();
    }
    public static void main(String[] args) {
        String infix = "A+B*C";
        System.out.println("Infix: " + infix);
        System.out.println("Postfix: " + convert(infix));
    }
}
```

Output:-



```
Infix: A+B*C
Postfix: ABC*+
```

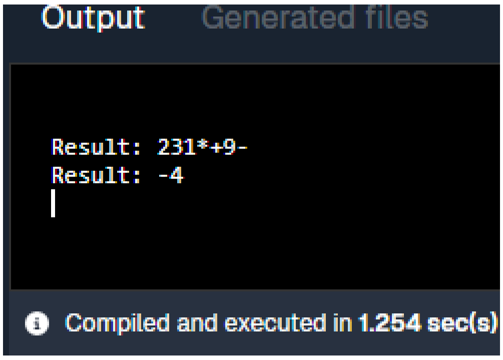
Compiled and executed in 1.218 sec(s)

17. Write a program to evaluate postfix expression.

Code:-

```
import java.util.Stack;
public class EvaluatePostfix {
    static int evaluate(String exp) {
        Stack<Integer> stack = new Stack<>();
        for (char c : exp.toCharArray()) {
            if (Character.isDigit(c)) stack.push(c - '0');
            else {
                int b = stack.pop(), a = stack.pop();
                switch (c) {
                    case '+': stack.push(a + b); break;
                    case '-': stack.push(a - b); break;
                    case '*': stack.push(a * b); break;
                    case '/': stack.push(a / b); break;
                }
            }
        }
        return stack.pop();
    }
    public static void main(String[] args) {
        String postfix = "231*+9-";
        System.out.println("Result: " +postfix);
        System.out.println("Result: " + evaluate(postfix));
    }
}
```

Output:-



```
Result: 231*+9-
Result: -4
```

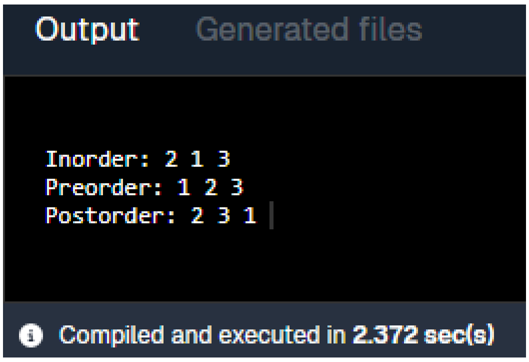
Compiled and executed in 1.254 sec(s)

18. Write a program to find out the preorder, inorder and postorder traversal of the tree.

Code:-

```
public class TreeTraversal {
    static class Node {
        int data;
        Node left, right;
        Node(int d) { data = d; }
    }
    static void inorder(Node root) {
        if (root != null) {
            inorder(root.left);
            System.out.print(root.data + " ");
            inorder(root.right);
        }
    }
    static void preorder(Node root) {
        if (root != null) {
            System.out.print(root.data + " ");
            preorder(root.left);
            preorder(root.right);
        }
    }
    static void postorder(Node root) {
        if (root != null) {
            postorder(root.left);
            postorder(root.right);
            System.out.print(root.data + " ");
        }
    }
    public static void main(String[] args) {
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        System.out.print("Inorder: ");
        inorder(root);
        System.out.print("\nPreorder: ");
        preorder(root);
        System.out.print("\nPostorder: ");
        postorder(root);
    }
}
```

Output:-



```
Inorder: 2 1 3
Preorder: 1 2 3
Postorder: 2 3 1
```

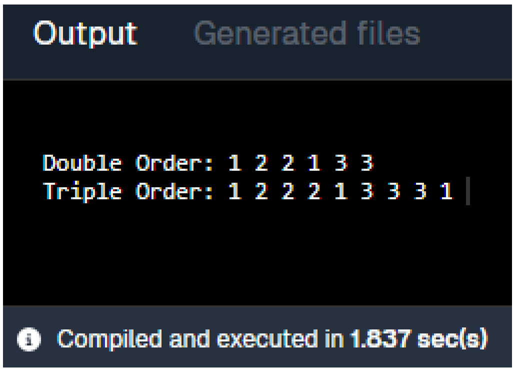
Compiled and executed in 2.372 sec(s)

19. Write a program to perform double-order traversal and triple-order traversal on the tree.

Code:-

```
public class DoubleTripleOrderTraversal {
    static class Node {
        int data;
        Node left, right;
        Node(int d) { data = d; }
    }

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

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

    public static void main(String[] args) {
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        System.out.print("Double Order: ");
        doubleOrder(root);
        System.out.print("\nTriple Order: ");
        tripleOrder(root);
    }
}
```

Output:-



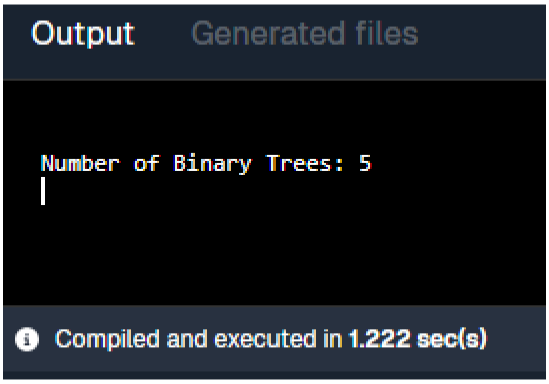
```
Double Order: 1 2 2 1 3 3
Triple Order: 1 2 2 2 1 3 3 3 1
```

20. Write a program to find the number of binary trees possible with given number of nodes.

Code:-

```
class NumberOfBinaryTrees {
    static int countTrees(int n) {
        int res = 1;
        for (int i = 0; i < n; ++i) {
            res = res * 2 * (2 * i + 1) / (i + 2);
        }
        return res;
    }

    public static void main(String[] args) {
        int n = 3;
        System.out.println("Number of Binary Trees: " + countTrees(n));
    }
}
```

Output:-



```
Number of Binary Trees: 5
```

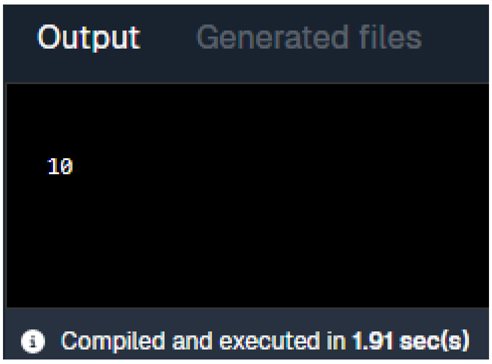
Compiled and executed in 1.222 sec(s)

21. Write a program to perform indirect recursion on the tree.

Code:-

```
public class IndirectRecursionTree {
    static class Node {
        int data;
        Node left, right;
        Node(int d) { data = d; }
    }
    static void A(Node root) {
        if (root != null) {
            System.out.print(root.data + " ");
            B(root.left);
        }
    }
    static void B(Node root) {
        if (root != null) {
            A(root.right);
        }
    }
    public static void main(String[] args) {
        Node root = new Node(10);
        root.left = new Node(20);
        root.right = new Node(30);
        A(root);
    }
}
```

Output:-



```
10
```

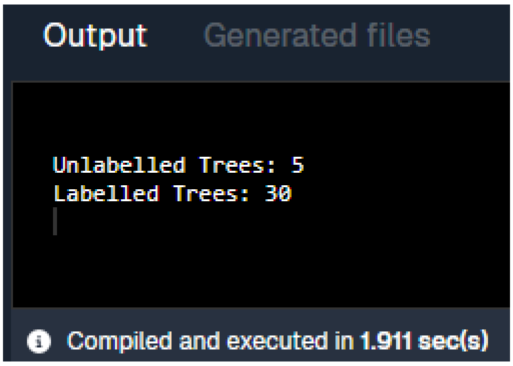
Compiled and executed in 1.91 sec(s)

22. Write a program to find out possible labelled and unlabeled binary trees with the given number of nodes.

Code:-

```
class BinaryTreePossibility {
    static int catalan(int n) {
        int res = 1;
        for (int i = 0; i < n; i++)
            res = res * 2 * (2 * i + 1) / (i + 2);
        return res;
    }
    static int labelled(int n) {
        int fact = 1;
        for (int i = 1; i <= n; i++) fact *= i;
        return catalan(n) * fact;
    }
    public static void main(String[] args) {
        int n = 3;
        System.out.println("Unlabelled Trees: " + catalan(n));
        System.out.println("Labelled Trees: " + labelled(n));
    }
}
```

Output:-



```
Unlabelled Trees: 5
Labelled Trees: 30
```

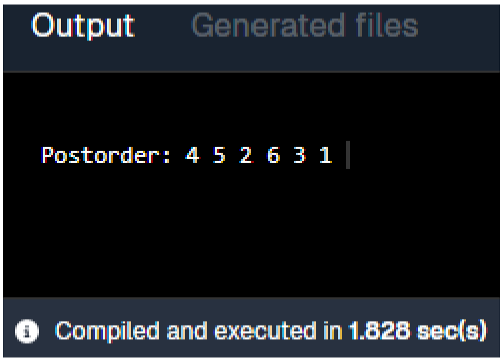
Compiled and executed in 1.911 sec(s)

23. Write a program to construct the unique binary tree using inorder and preorder traversal and hence find postorder.

Code:-

```
public class ConstructTree {
    static class Node {
        int data;
        Node left, right;
        Node(int d) { data = d; }
    }
    static int preIndex = 0;
    static Node buildTree(int[] in, int[] pre, int inStart, int inEnd) {
        if (inStart > inEnd) return null;
        Node node = new Node(pre[preIndex++]);
        if (inStart == inEnd) return node;
        int inIndex = search(in, inStart, inEnd, node.data);
        node.left = buildTree(in, pre, inStart, inIndex - 1);
        node.right = buildTree(in, pre, inIndex + 1, inEnd);
        return node;
    }
    static int search(int[] arr, int start, int end, int value) {
        for (int i = start; i <= end; i++)
            if (arr[i] == value) return i;
        return -1;
    }
    static void postorder(Node node) {
        if (node != null) {
            postorder(node.left);
            postorder(node.right);
            System.out.print(node.data + " ");
        }
    }
    public static void main(String[] args) {
        int[] inorder = {4, 2, 5, 1, 6, 3};
        int[] preorder = {1, 2, 4, 5, 3, 6};
        Node root = buildTree(inorder, preorder, 0, inorder.length - 1);
        System.out.print("Postorder: ");
        postorder(root);
    }
}
```

Output:-



```
Postorder: 4 5 2 6 3 1
```

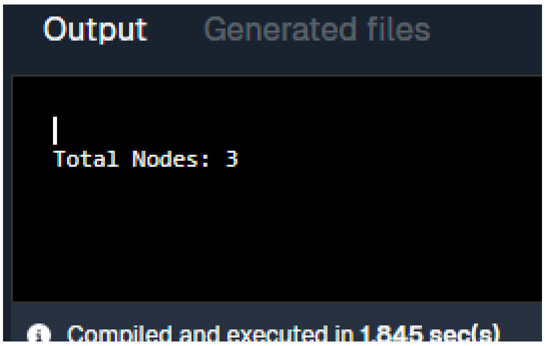
Compiled and executed in 1.828 sec(s)

24. Write a recursive program to count the total number of nodes in the tree.

Code:-

```
public class CountNodes {
    static class Node {
        int data;
        Node left, right;
        Node(int d) { data = d; }
    }
    static int count(Node root) {
        if (root == null) return 0;
        return 1 + count(root.left) + count(root.right);
    }
    public static void main(String[] args) {
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        System.out.println("Total Nodes: " + count(root));
    }
}
```

Output:-



```
Total Nodes: 3
```

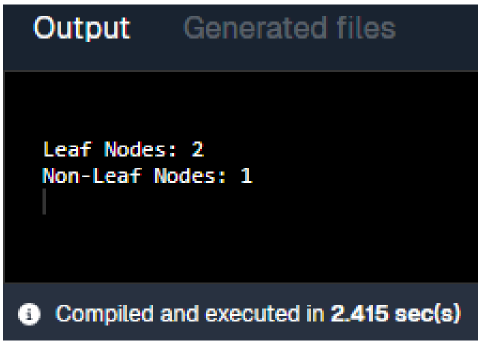
Compiled and executed in 1.845 sec(s)

25. Write a recursive program to count the number of the leaf or non-leaf nodes of the tree.

Code:-

```
public class LeafNonLeafCount {
    static class Node {
        int data;
        Node left, right;
        Node(int d) { data = d; }
    }
    static int leafCount(Node root) {
        if (root == null) return 0;
        if (root.left == null && root.right == null) return 1;
        return leafCount(root.left) + leafCount(root.right);
    }
    static int nonLeafCount(Node root) {
        if (root == null || (root.left == null && root.right == null)) return 0;
        return 1 + nonLeafCount(root.left) + nonLeafCount(root.right);
    }
    public static void main(String[] args) {
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        System.out.println("Leaf Nodes: " + leafCount(root));
        System.out.println("Non-Leaf Nodes: " + nonLeafCount(root));
    }
}
```

Output:-



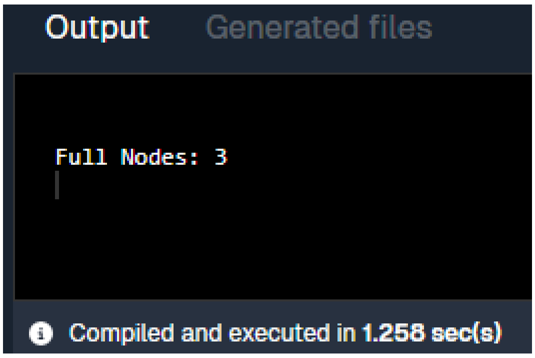
```
Leaf Nodes: 2
Non-Leaf Nodes: 1
```

26. Write a recursive program to count the number of full nodes of the tree (Full Nodes are nodes which has both left and right children as non-empty).

Code:-

```
public class FullNodesCount {
    static class Node {
        int data;
        Node left, right;
        Node(int d) { data = d; }
    }
    static int countFull(Node root) {
        if (root == null) return 0;
        int count = countFull(root.left) + countFull(root.right);
        if (root.left != null && root.right != null) count++;
        return count;
    }
    public static void main(String[] args) {
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.right.right = new Node(5);
        root.right.left = new Node(7);
        root.left.right = new Node(9);
        System.out.println("Full Nodes: " + countFull(root));
    }
}
```

Output:-



```
Full Nodes: 3
```

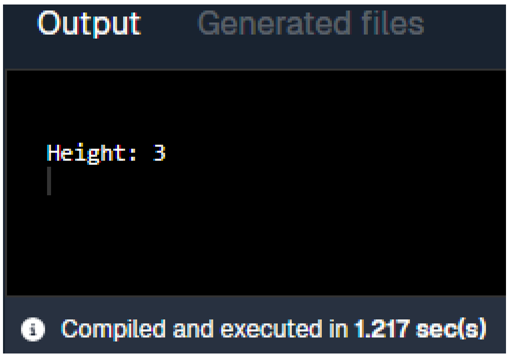
Compiled and executed in 1.258 sec(s)

27. Write a recursive program to find the height of the tree.

Code:-

```
class TreeHeight {
    static class Node {
        int data;
        Node left, right;
        Node(int d) { data = d; }
    }

    static int height(Node root) {
        if (root == null) return 0;
        return 1 + Math.max(height(root.left), height(root.right));
    }

    public static void main(String[] args) {
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        System.out.println("Height: " + height(root));
    }
}
```

Output:-



```
Height: 3
```

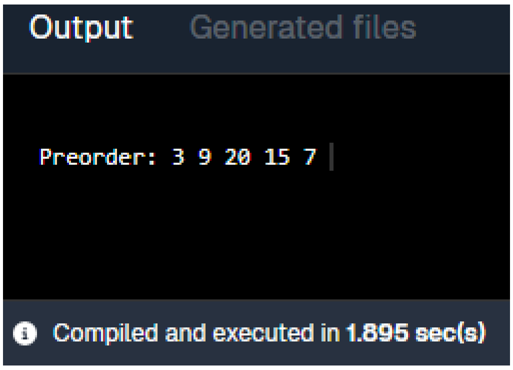
Compiled and executed in 1.217 sec(s)

28. Write a program to construct BST using inorder and postorder traversal and hence find preorder.

Code:-

```
public class BSTFromInPost {
    static class Node {
        int data;
        Node left, right;
        Node(int d) { data = d; }
    }
    static int postIndex;
    static Node buildTree(int[] inorder, int[] postorder, int inStart, int inEnd) {
        if (inStart > inEnd) return null;
        Node node = new Node(postorder[postIndex--]);
        if (inStart == inEnd) return node;
        int inIndex = search(inorder, inStart, inEnd, node.data);
        node.right = buildTree(inorder, postorder, inIndex + 1, inEnd);
        node.left = buildTree(inorder, postorder, inStart, inIndex - 1);
        return node;
    }
    static int search(int[] arr, int start, int end, int val) {
        for (int i = start; i <= end; i++)
            if (arr[i] == val) return i;
        return -1;
    }
    static void preorder(Node root) {
        if (root != null) {
            System.out.print(root.data + " ");
            preorder(root.left);
            preorder(root.right);
        }
    }
    public static void main(String[] args) {
        int[] inorder = {9, 3, 15, 20, 7};
        int[] postorder = {9, 15, 7, 20, 3};
        postIndex = postorder.length - 1;
        Node root = buildTree(inorder, postorder, 0, inorder.length - 1);
        System.out.print("Preorder: ");
        preorder(root);
    }
}
```

Output:-



```
Preorder: 3 9 20 15 7
```

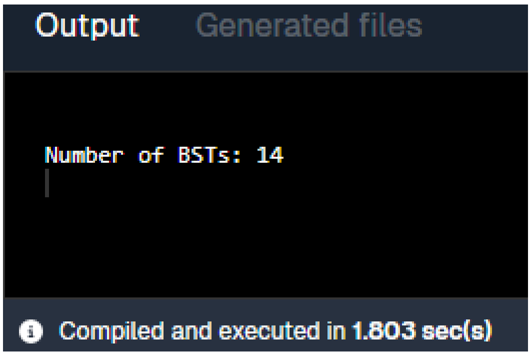
Compiled and executed in 1.895 sec(s)

29. Write a program to find how many BSTs are possible with given distinct keys.

Code:-

```
public class PossibleBST {
    static int catalan(int n) {
        int res = 1;
        for (int i = 0; i < n; ++i)
            res = res * 2 * (2 * i + 1) / (i + 2);
        return res;
    }

    public static void main(String[] args) {
        int keys = 4;
        System.out.println("Number of BSTs: " + catalan(keys));
    }
}
```

Output:-



```
Number of BSTs: 14
```

Compiled and executed in 1.803 sec(s)

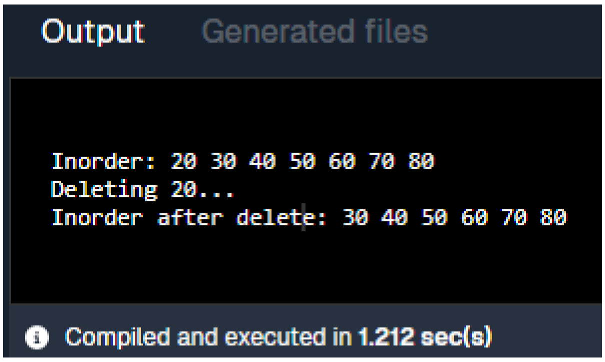
30. Write a program to find out the postorder, preorder and inorder traversal on constructed bst and then perform delete operation on the tree and again perform inorder traversal.

Code:-

```
public class BSTDelete {
    static class Node {
        int key;
        Node left, right;
        Node(int d) { key = d; }
    }
    static Node insert(Node root, int key) {
        if (root == null) return new Node(key);
        if (key < root.key) root.left = insert(root.left, key);
        else root.right = insert(root.right, key);
        return root;
    }
    static Node delete(Node root, int key) {
        if (root == null) return root;
        if (key < root.key) root.left = delete(root.left, key);
        else if (key > root.key) root.right = delete(root.right, key);
        else {
            if (root.left == null) return root.right;
            if (root.right == null) return root.left;
            root.key = minValue(root.right);
            root.right = delete(root.right, root.key);
        }
        return root;
    }
    static int minValue(Node root) {
        int minv = root.key;
        while (root.left != null) {
            minv = root.left.key;
            root = root.left;
        }
        return minv;
    }
    static void inorder(Node root) {
        if (root != null) {
            inorder(root.left);
            System.out.print(root.key + " ");
            inorder(root.right);
        }
    }
    static void preorder(Node root) {
        if (root != null) {
            System.out.print(root.key + " ");
            preorder(root.left);
            preorder(root.right);
        }
    }
    static void postorder(Node root) {
        if (root != null) {
            postorder(root.left);
            postorder(root.right);
            System.out.print(root.key + " ");
        }
    }
    public static void main(String[] args) {
        Node root = null;
        int[] keys = {50, 30, 20, 40, 70, 60, 80};
```

```
        for (int key : keys) root = insert(root, key);
        System.out.print("Inorder: ");
        inorder(root);
        System.out.println("\nDeleting 20...");
        root = delete(root, 20);
        System.out.print("Inorder after delete: ");
        inorder(root);
    }
}
```

Output:-



```
Inorder: 20 30 40 50 60 70 80
Deleting 20...
Inorder after delete: 30 40 50 60 70 80
```

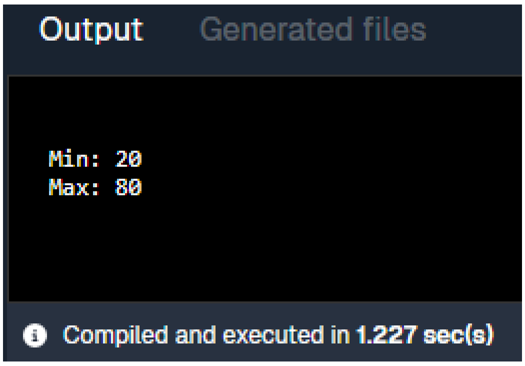
Compiled and executed in 1.212 sec(s)

31. Write a program to find the minimum and maximum key values from BSTs.

Code:-

```
public class MinMaxBST {
    static class Node {
        int key;
        Node left, right;
        Node(int d) { key = d; }
    }
    static int findMin(Node root) {
        while (root.left != null) root = root.left;
        return root.key;
    }
    static int findMax(Node root) {
        while (root.right != null) root = root.right;
        return root.key;
    }
    public static void main(String[] args) {
        Node root = new Node(50);
        root.left = new Node(30);
        root.right = new Node(70);
        root.left.left = new Node(20);
        root.right.right = new Node(80);
        System.out.println("Min: " + findMin(root));
        System.out.println("Max: " + findMax(root));
    }
}
```

Output:-



```
Min: 20
Max: 80
```

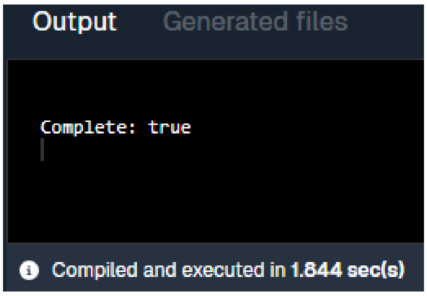
Compiled and executed in 1.227 sec(s)

32. Write a recursive program to check whether given tree is complete tree or not.

Code:-

```
public class CompleteBinaryTree {
    static class Node {
        int data;
        Node left, right;
        Node(int d) { data = d; }
    }
    static int countNodes(Node root) {
        if (root == null) return 0;
        return 1 + countNodes(root.left) + countNodes(root.right);
    }
    static boolean isComplete(Node root, int index, int n) {
        if (root == null) return true;
        if (index >= n) return false;
        return isComplete(root.left, 2 * index + 1, n) && isComplete(root.right,
2 * index + 2, n);
    }
    public static void main(String[] args) {
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        int nodeCount = countNodes(root);

        System.out.println("Complete: " + isComplete(root, 0, nodeCount));
    }
}
```

Output:-



```
Complete: true
```

Compiled and executed in 1.844 sec(s)

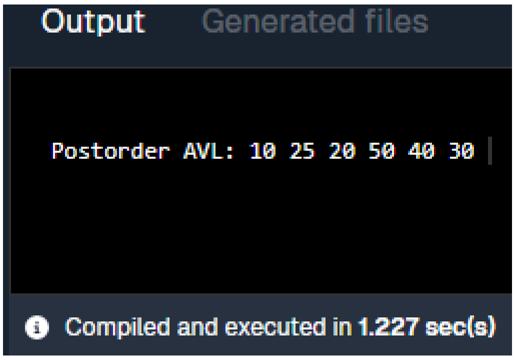
33. Write a program to construct an AVL tree and perform postorder traversal.

Code:-

```
class AVLTree {
    static class Node {
        int key, height;
        Node left, right;
        Node(int d) { key = d; height = 1; }
    }
    static int height(Node N) {
        return (N == null) ? 0 : N.height;
    }
    static int getBalance(Node N) {
        return (N == null) ? 0 : height(N.left) - height(N.right);
    }
    static Node rightRotate(Node y) {
        Node x = y.left;
        Node T2 = x.right;
        x.right = y;
        y.left = T2;
        y.height = 1 + Math.max(height(y.left), height(y.right));
        x.height = 1 + Math.max(height(x.left), height(x.right));
        return x;
    }
    static Node leftRotate(Node x) {
        Node y = x.right;
        Node T2 = y.left;
        y.left = x;
        x.right = T2;
        x.height = 1 + Math.max(height(x.left), height(x.right));
        y.height = 1 + Math.max(height(y.left), height(y.right));
        return y;
    }
    static Node insert(Node node, int key) {
        if (node == null) return new Node(key);
        if (key < node.key) node.left = insert(node.left, key);
        else if (key > node.key) node.right = insert(node.right, key);
        else return node;
        node.height = 1 + Math.max(height(node.left), height(node.right));
        int balance = getBalance(node);
        if (balance > 1 && key < node.left.key) return rightRotate(node);
        if (balance < -1 && key > node.right.key) return leftRotate(node);
        if (balance > 1 && key > node.left.key) {
            node.left = leftRotate(node.left);
            return rightRotate(node);
        }
        if (balance < -1 && key < node.right.key) {
            node.right = rightRotate(node.right);
            return leftRotate(node);
        }
        return node;
    }
    static void postorder(Node root) {
        if (root != null) {
            postorder(root.left);
            postorder(root.right);
            System.out.print(root.key + " ");
        }
    }
```

```
    public static void main(String[] args) {
        Node root = null;
        int[] keys = {10, 20, 30, 40, 50, 25};
        for (int key : keys) root = insert(root, key);
        System.out.print("Postorder AVL: ");
        postorder(root);
    }
}
```

Output:-



```
Postorder AVL: 10 25 20 50 40 30
```

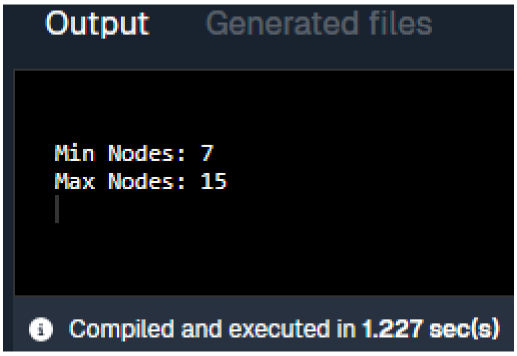
Compiled and executed in 1.227 sec(s)

34. Write a program to find the minimum and maximum nodes in an AVL tree of given height.

Code:-

```
public class AVLMinMaxNodes {
    static int minNodes(int h) {
        if (h == 0) return 1;
        if (h == 1) return 2;
        return 1 + minNodes(h - 1) + minNodes(h - 2);
    }

    static int maxNodes(int h) {
        return (int) Math.pow(2, h + 1) - 1;
    }

    public static void main(String[] args) {
        int height = 3;
        System.out.println("Min Nodes: " + minNodes(height));
        System.out.println("Max Nodes: " + maxNodes(height));
    }
}
```

Output:-



```
Min Nodes: 7
Max Nodes: 15
```

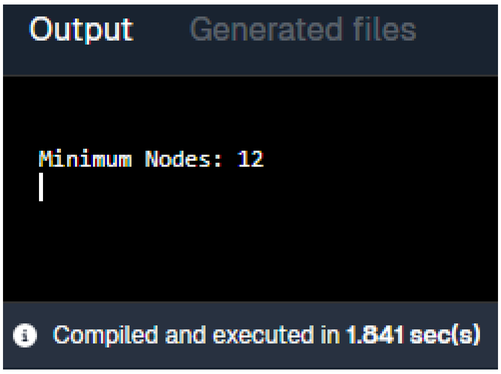
Compiled and executed in 1.227 sec(s)

35. Write a program to find the minimum number of nodes on a size-balanced tree.

Code:-

```
public class SizeBalancedTree {
    static int minNodes(int h) {
        if (h == 0) return 1;
        if (h == 1) return 2;
        return 1 + minNodes(h - 1) + minNodes(h - 2);
    }

    public static void main(String[] args) {
        int height = 4;
        System.out.println("Minimum Nodes: " + minNodes(height));
    }
}
```

Output:-



```
Minimum Nodes: 12
```

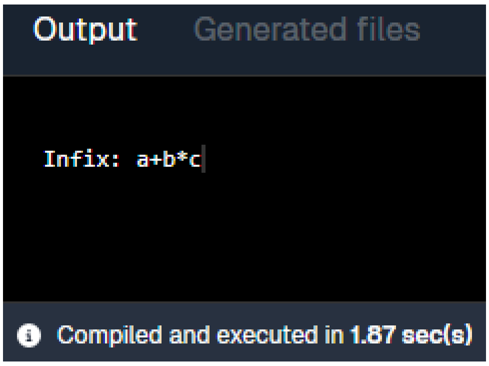
Compiled and executed in 1.841 sec(s)

36. Write a program to implement a tree for a given infix expression.

Code:-

```
class ExpressionTree {
    static class Node {
        char value;
        Node left, right;
        Node(char v) { value = v; }
    }
    static Node construct(String expr) {
        java.util.Stack<Node> stack = new java.util.Stack<>();
        for (char ch : expr.toCharArray()) {
            if (Character.isLetterOrDigit(ch)) stack.push(new Node(ch));
            else {
                Node node = new Node(ch);
                node.right = stack.pop();
                node.left = stack.pop();
                stack.push(node);
            }
        }
        return stack.peek();
    }
    static void inorder(Node root) {
        if (root != null) {
            inorder(root.left);
            System.out.print(root.value);
            inorder(root.right);
        }
    }
    public static void main(String[] args) {
        String postfix = "ab+c*";
        Node root = construct(postfix);
        System.out.print("Infix: ");
        inorder(root);
    }
}
```

Output:-



```
Infix: a+b*c
```

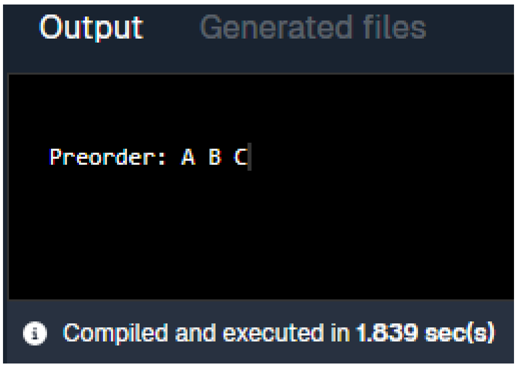
Compiled and executed in 1.87 sec(s)

37. Write a program to draw a tree for a given nested tree representation expression.

Code:-

```
public class NestedTree {
    static class Node {
        char data;
        Node left, right;
        Node(char d) { data = d; }
    }
    static Node buildTree(String expr, int[] index) {
        if (index[0] >= expr.length()) return null;
        char ch = expr.charAt(index[0]);
        if (ch == '(') {
            index[0]++;
            char nodeData = expr.charAt(index[0]++);
            Node root = new Node(nodeData);
            root.left = buildTree(expr, index);
            index[0]++;
            root.right = buildTree(expr, index);
            index[0]++;
            return root;
        }
        return null;
    }
    static void preorder(Node root) {
        if (root != null) {
            System.out.print(root.data + " ");
            preorder(root.left);
            preorder(root.right);
        }
    }
    public static void main(String[] args) {
        String expr = "(A(B,C))";
        int[] index = {0};
        Node root = buildTree(expr, index);
        System.out.print("Preorder: ");
        preorder(root);
    }
}
```

Output:-



```
Preorder: A B C
```

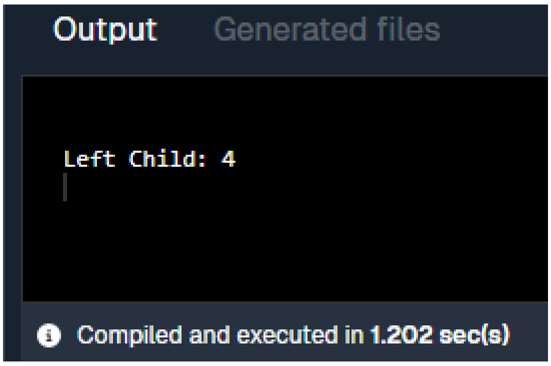
Compiled and executed in **1.839 sec(s)**

38. Write a program to find the left child of kth element from the given array representation tree.

Code:-

```
public class LeftChildArrayTree {
    public static void main(String[] args) {
        int[] tree = {1, 2, 3, 4, 5, 6, 7};
        int k = 1;
        int leftChildIdx = 2 * k + 1;
        if (leftChildIdx < tree.length)
            System.out.println("Left Child: " + tree[leftChildIdx]);
        else
            System.out.println("No Left Child");
    }
}
```

Output:-



```
Left Child: 4
```

Compiled and executed in 1.202 sec(s)

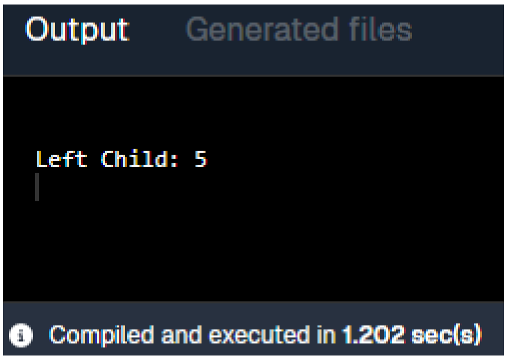
39. Write a program to find the left child of kth element from the given leftmost child right sibling representation tree.

Code:-

```
public class LeftmostChildTree {
    static class Node {
        int data;
        Node leftChild, rightSibling;
        Node(int d) { data = d; }

    static Node findKth(Node root, int k, int[] count) {
        if (root == null) return null;
        if (count[0] == k) return root;
        count[0]++;
        Node res = findKth(root.leftChild, k, count);
        if (res == null) res = findKth(root.rightSibling, k, count);
        return res;
    }

    public static void main(String[] args) {
        Node root = new Node(1);
        root.leftChild = new Node(2);
        root.leftChild.rightSibling = new Node(3);
        root.leftChild.rightSibling.rightSibling = new Node(4);
        root.leftChild.leftChild = new Node(5);
        int k = 1;

        Node kth = findKth(root, k, new int[]{0});
        if (kth != null && kth.leftChild != null)
            System.out.println("Left Child: " + kth.leftChild.data);
        else
```

```
            System.out.println("No Left Child");
    }
}
```

Output:-



```
Left Child: 5
```

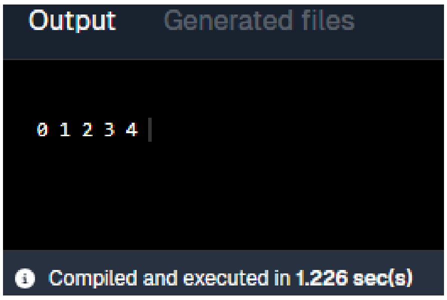
Compiled and executed in 1.202 sec(s)

40. Write a program for breadth first traversal on graph.

Code:-

```
import java.util.*;
public class GraphBFS {
    static void bfs(int v, List<List<Integer>> adj) {
        boolean[] visited = new boolean[v];
        Queue<Integer> q = new LinkedList<>();
        visited[0] = true;
        q.add(0);
        while (!q.isEmpty()) {
            int node = q.poll();
            System.out.print(node + " ");
            for (int neigh : adj.get(node)) {
                if (!visited[neigh]) {
                    visited[neigh] = true;
                    q.add(neigh);
                }
            }
        }
    }
    public static void main(String[] args) {
        int v = 5;
        List<List<Integer>> adj = new ArrayList<>();
        for (int i = 0; i < v; i++) adj.add(new ArrayList<>());
        adj.get(0).add(1); adj.get(0).add(2);
        adj.get(1).add(3); adj.get(2).add(4);
        bfs(v, adj);
    }
}
```

Output:



```
0 1 2 3 4
```

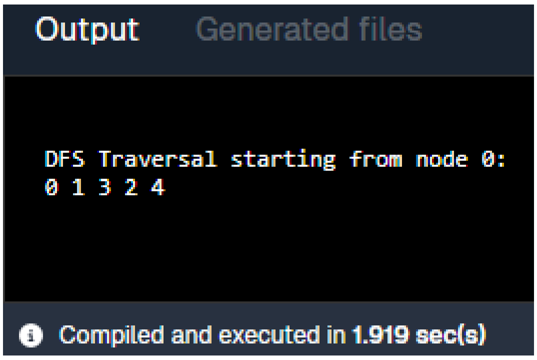
Compiled and executed in 1.226 sec(s)

41. Write a program for depth first traversal on graph.

Code:-

```
import java.util.*;
public class GraphDFS {
    static void dfs(int node, boolean[] visited, List<List<Integer>> adj) {
        visited[node] = true;
        System.out.print(node + " ");
        for (int neigh : adj.get(node))
            if (!visited[neigh]) dfs(neigh, visited, adj);
    }
    public static void main(String[] args) {
        int v = 5;
        List<List<Integer>> adj = new ArrayList<>();
        for (int i = 0; i < v; i++) adj.add(new ArrayList<>());
        adj.get(0).add(1); adj.get(0).add(2);
        adj.get(1).add(3); adj.get(2).add(4);
        boolean[] visited = new boolean[v];
        System.out.println("DFS Traversal starting from node 0:");
        dfs(0, visited, adj);
        System.out.println();
    }
}
```

Output:-



```
DFS Traversal starting from node 0:
0 1 3 2 4
```

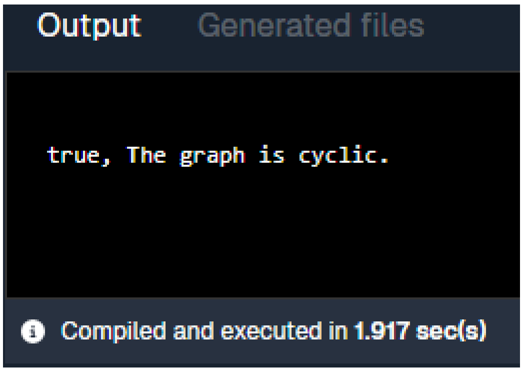
Compiled and executed in 1.919 sec(s)

42. Write a program to check whether there is a cycle in a given directed graph or not.

Code:-

```
import java.util.*;


public class DirectedCycle {
    static boolean isCyclic(int v, List<List<Integer>> adj) {
        boolean[] visited = new boolean[v];
        boolean[] recStack = new boolean[v];
        for (int i = 0; i < v; i++)
            if (dfsCycle(i, visited, recStack, adj)) return true;
        return false;
    }


    static boolean dfsCycle(int node, boolean[] visited, boolean[] recStack, List<List<Integer>> adj) {
        if (recStack[node]) return true;
        if (visited[node]) return false;
        visited[node] = recStack[node] = true;
        for (int neigh : adj.get(node))
            if (dfsCycle(neigh, visited, recStack, adj)) return true;
        recStack[node] = false;
        return false;
    }



public static void main(String[] args) {
        int v = 4;
        List<List<Integer>> adj = new ArrayList<>();
        for (int i = 0; i < v; i++) adj.add(new ArrayList<>());
        adj.get(0).add(1); adj.get(1).add(2); adj.get(2).add(0);
        System.out.println(isCyclic(v, adj)+", The graph is cyclic.");
    }
}
```

Output:-



```
true, The graph is cyclic.
```

Compiled and executed in 1.917 sec(s)

Searching and Sorting

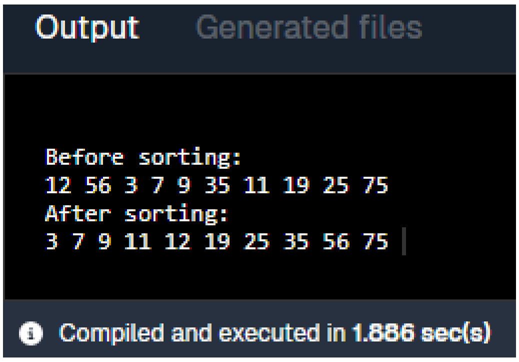
Use this data for all the following programs: 12, 56, 3, 7, 9, 35, 11, 19, 25, 75

43. Write a program to sort given elements using insertion sort method.

Code:-

```
public class InsertionSort {
    public static void main(String[] args) {
        int[] arr = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
        System.out.println("Before sorting:");
        for (int n : arr) System.out.print(n + " ");
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i], j = i - 1;
            while (j >= 0 && arr[j] > key)
                arr[j + 1] = arr[j--];
            arr[j + 1] = key;
        }
        System.out.println("\nAfter sorting:");
        for (int n : arr) System.out.print(n + " ");
    }
}
```

Output:-



```
Before sorting:
12 56 3 7 9 35 11 19 25 75
After sorting:
3 7 9 11 12 19 25 35 56 75
```

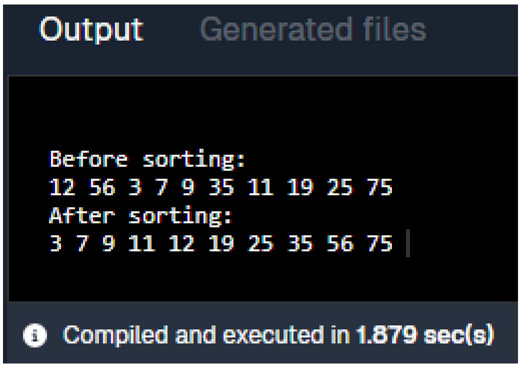
Compiled and executed in 1.886 sec(s)

44. Write a program to sort given elements using bubble sort method.

Code:-

```
public class BubbleSort {
    public static void main(String[] args) {
        int[] arr = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
        System.out.println("Before sorting:");
        for (int x : arr) System.out.print(x + " ");
        int n = arr.length;
        for (int i = 0; i < n - 1; i++)
            for (int j = 0; j < n - i - 1; j++)
                if (arr[j] > arr[j + 1]) {
                    int t = arr[j]; arr[j] = arr[j + 1]; arr[j + 1] = t;
                }

        System.out.println("\nAfter sorting:");
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Output:-



```
Before sorting:
12 56 3 7 9 35 11 19 25 75
After sorting:
3 7 9 11 12 19 25 35 56 75
```

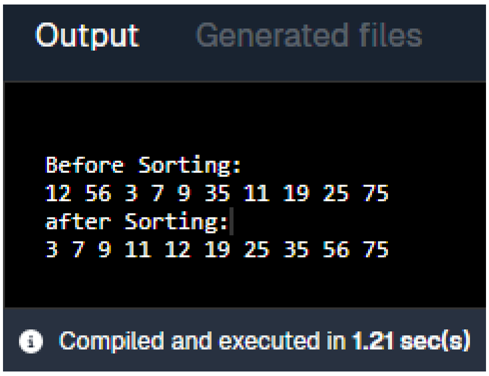
Compiled and executed in 1.879 sec(s)

45. Write a program to sort given elements using bucket sort method.

Code:-

```
import java.util.*;


public class BucketSort {
    public static void main(String[] args) {
        int[] arr = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
        System.out.println("Before Sorting:");
        for (int num : arr) System.out.print(num + " ");
        int max = Arrays.stream(arr).max().getAsInt();
        List<List<Integer>> buckets = new ArrayList<>();
        for (int i = 0; i <= max/10; i++) buckets.add(new ArrayList<>());
        for (int num : arr) buckets.get(num/10).add(num);
        for (List<Integer> bucket : buckets) Collections.sort(bucket);
        System.out.println("\nafter Sorting:");
        for (List<Integer> bucket : buckets)
            for (int num : bucket) System.out.print(num + " ");
    }
}
```

Output:-



```
Before Sorting:
12 56 3 7 9 35 11 19 25 75
after Sorting:
3 7 9 11 12 19 25 35 56 75
```

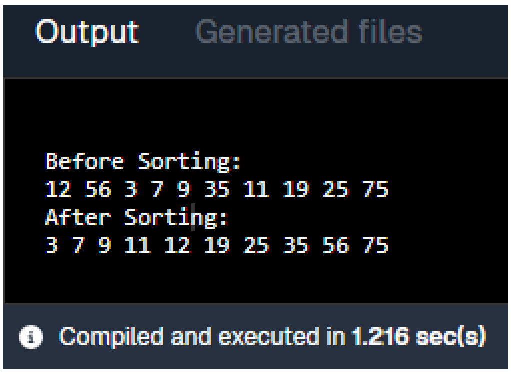
Compiled and executed in 1.21 sec(s)

46. Write a program to sort given elements using merge sort method.

Code:-

```
public class MergeSort {
    static void merge(int[] arr, int l, int m, int r) {
        int[] left = java.util.Arrays.copyOfRange(arr, l, m + 1);
        int[] right = java.util.Arrays.copyOfRange(arr, m + 1, r + 1);
        int i = 0, j = 0, k = l;
        while (i < left.length && j < right.length)
            arr[k++] = (left[i] <= right[j]) ? left[i++] : right[j++];
        while (i < left.length) arr[k++] = left[i++];
        while (j < right.length) arr[k++] = right[j++];
    }

    static void mergeSort(int[] arr, int l, int r) {
        if (l < r) {
            int m = (l + r) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }



    public static void main(String[] args) {
        int[] arr = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
        System.out.println("Before Sorting: ");
        for (int x : arr) System.out.print(x + " ");
        mergeSort(arr, 0, arr.length - 1);
        System.out.println("\nAfter Sorting:");
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Output:-



```
Before Sorting:
12 56 3 7 9 35 11 19 25 75
After Sorting:
3 7 9 11 12 19 25 35 56 75
```

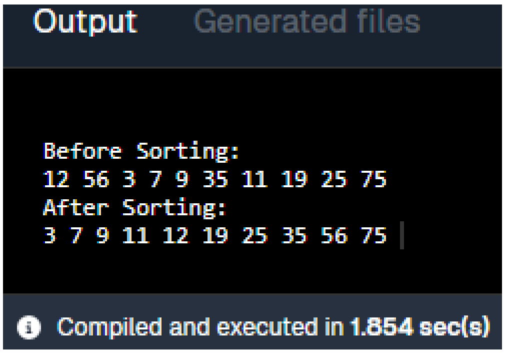
Compiled and executed in 1.216 sec(s)

47. Write a program to sort given elements using quick sort method.

Code:-

```
public class QuickSort {
    static int partition(int[] arr, int low, int high) {
        int pivot = arr[high], i = low - 1;
        for (int j = low; j < high; j++)
            if (arr[j] < pivot)
                { int t = arr[++i]; arr[i] = arr[j]; arr[j] = t; }
        int t = arr[i + 1]; arr[i + 1] = arr[high]; arr[high] = t;
        return i + 1;
    }
    static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int p = partition(arr, low, high);
            quickSort(arr, low, p - 1);
            quickSort(arr, p + 1, high);
        }
    }
    public static void main(String[] args) {
        int[] arr = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
        System.out.println("Before Sorting: ");
        for (int x : arr) System.out.print(x + " ");
        quickSort(arr, 0, arr.length - 1);
        System.out.println("\nAfter Sorting: ");
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Output:-



```
Before Sorting:
12 56 3 7 9 35 11 19 25 75
After Sorting:
3 7 9 11 12 19 25 35 56 75
```

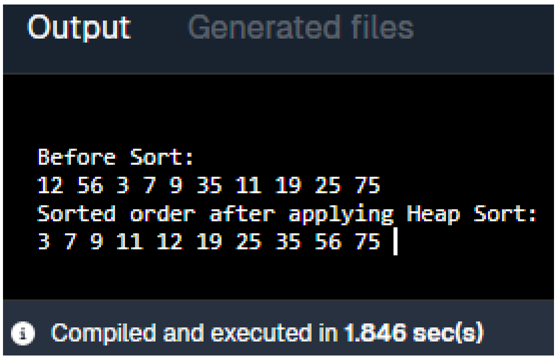
Compiled and executed in 1.854 sec(s)

48. Write a program to sort given elements using heap sort method.

Code:-

```
public class HeapSort {
    static void heapify(int[] arr, int n, int i) {
        int largest = i, l = 2*i+1, r = 2*i+2;
        if (l < n && arr[l] > arr[largest]) largest = l;
        if (r < n && arr[r] > arr[largest]) largest = r;
        if (largest != i) {
            int t = arr[i]; arr[i] = arr[largest]; arr[largest] = t;
            heapify(arr, n, largest);
        }
    }
    public static void main(String[] args) {
        int[] arr = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
        System.out.println("Before Sort: ");
        for(int num: arr) System.out.print(num + " ");
        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 t = arr[0]; arr[0] = arr[i]; arr[i] = t;
            heapify(arr, i, 0);
        }
        System.out.println("\n Sorted order after applying Heap Sort:");
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Output:-



```
Before Sort:
12 56 3 7 9 35 11 19 25 75
Sorted order after applying Heap Sort:
3 7 9 11 12 19 25 35 56 75
```

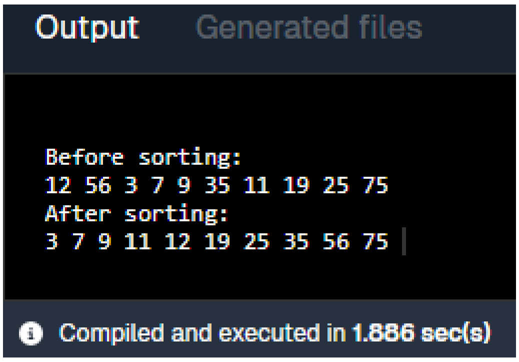
Compiled and executed in 1.846 sec(s)

49. Write a program to sort given elements using insertion sort method.

Code:-

```
public class InsertionSort {
    public static void main(String[] args) {
        int[] arr = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};
        System.out.println("Before sorting:");
        for (int n : arr) System.out.print(n + " ");
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i], j = i - 1;
            while (j >= 0 && arr[j] > key)
                arr[j + 1] = arr[j--];
            arr[j + 1] = key;
        }
        System.out.println("\nAfter sorting:");
        for (int n : arr) System.out.print(n + " ");
    }
}
```

Output:-



```
Before sorting:
12 56 3 7 9 35 11 19 25 75
After sorting:
3 7 9 11 12 19 25 35 56 75
```

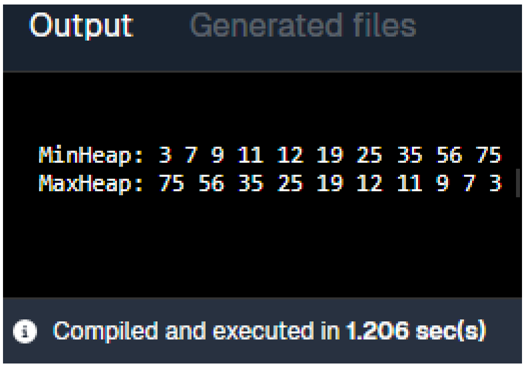
Compiled and executed in 1.886 sec(s)

50. Write a program to construct min heap and max heap.

Code:-

```
import java.util.*;

public class MinMaxHeap {

    public static void main(String[] args) {

        int[] arr = {12, 56, 3, 7, 9, 35, 11, 19, 25, 75};

        PriorityQueue<Integer> minHeap = new PriorityQueue<>();

        PriorityQueue<Integer> maxHeap = new
PriorityQueue<>(Collections.reverseOrder());

        for (int n : arr) {

            minHeap.add(n);

            maxHeap.add(n);

        }

        System.out.print("MinHeap: ");

        while (!minHeap.isEmpty()) {

            System.out.print(minHeap.poll() + " ");

        }

        System.out.print("\nMaxHeap: ");

        while (!maxHeap.isEmpty()) {

            System.out.print(maxHeap.poll() + " ");

        }

    }

}
```

Output:-



```
MinHeap: 3 7 9 11 12 19 25 35 56 75
MaxHeap: 75 56 35 25 19 12 11 9 7 3
```

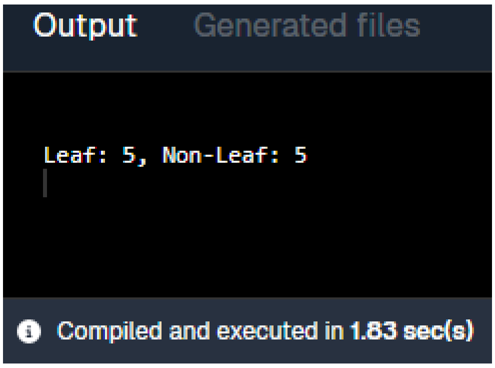
Compiled and executed in 1.206 sec(s)

51. Write a program to find the number of leaf and non-leaf nodes of a max heap.

Code:-

```
public class HeapLeafNonLeaf {
    public static void main(String[] args) {
        int[] heap = {75, 56, 35, 19, 25, 3, 11, 7, 9, 12};
        int n = heap.length, leaf = 0, nonLeaf = 0;
        for (int i = 0; i < n; i++)
            if (2*i+1 >= n) leaf++;
            else nonLeaf++;
        System.out.println("Leaf: " + leaf + ", Non-Leaf: " + nonLeaf);
    }
}
```

Output:-



```
Leaf: 5, Non-Leaf: 5
```

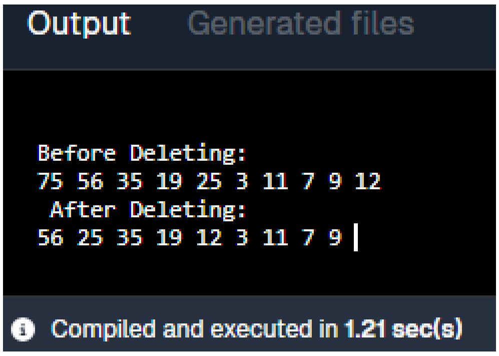
Compiled and executed in 1.83 sec(s)

52. Write a program to delete maximum value from a max heap and then reheapify.

Code:-

```
public class DeleteMaxHeap {
    static void heapify(int[] heap, int n, int i) {
        int largest = i, l = 2*i+1, r = 2*i+2;
        if (l<n && heap[l]>heap[largest]) largest=l;
        if (r<n && heap[r]>heap[largest]) largest=r;
        if (largest != i) {
            int t = heap[i]; heap[i] = heap[largest]; heap[largest] = t;
            heapify(heap, n, largest);
        }
    }
    public static void main(String[] args) {
        int[] heap = {75, 56, 35, 19, 25, 3, 11, 7, 9, 12};
        System.out.println("Before Deleting: ");
        for(int n: heap) System.out.print(n+" ");
        int n = heap.length;
        heap[0] = heap[n-1];
        n--;
        heapify(heap, n, 0);
         System.out.println("\n After Deleting: ");
        for (int i=0; i<n; i++) System.out.print(heap[i] + " ");
    }
}
```

Output:-



```
Before Deleting:
75 56 35 19 25 3 11 7 9 12
 After Deleting:
56 25 35 19 12 3 11 7 9
```

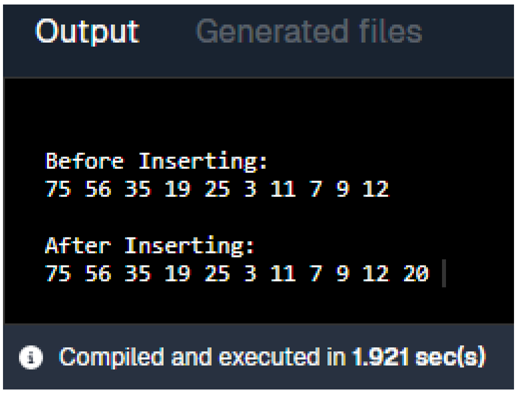
Compiled and executed in 1.21 sec(s)

53. Write a program to insert 20 in max heap.

Code:-

```
import java.util.*;
public class InsertMaxHeap {
    static void heapifyUp(int[] heap, int i) {
        while (i > 0 && heap[(i - 1) / 2] < heap[i]) {
            int temp = heap[i];
            heap[i] = heap[(i - 1) / 2];
            heap[(i - 1) / 2] = temp;
            i = (i - 1) / 2;
        }
    }
    public static void main(String[] args) {
        int[] heap = {75, 56, 35, 19, 25, 3, 11, 7, 9, 12, 0};
        System.out.println("Before Inserting:");
        for (int i = 0; i < 10; i++) System.out.print(heap[i] + " ");
        int n = 10;
        heap[n] = 20;
        heapifyUp(heap, n);
        System.out.println("\n\nAfter Inserting:");
        for (int i = 0; i <= n; i++) System.out.print(heap[i] + " ");
    }
}
```

Output:-



```
Before Inserting:
75 56 35 19 25 3 11 7 9 12

After Inserting:
75 56 35 19 25 3 11 7 9 12 20
```

Compiled and executed in 1.921 sec(s)

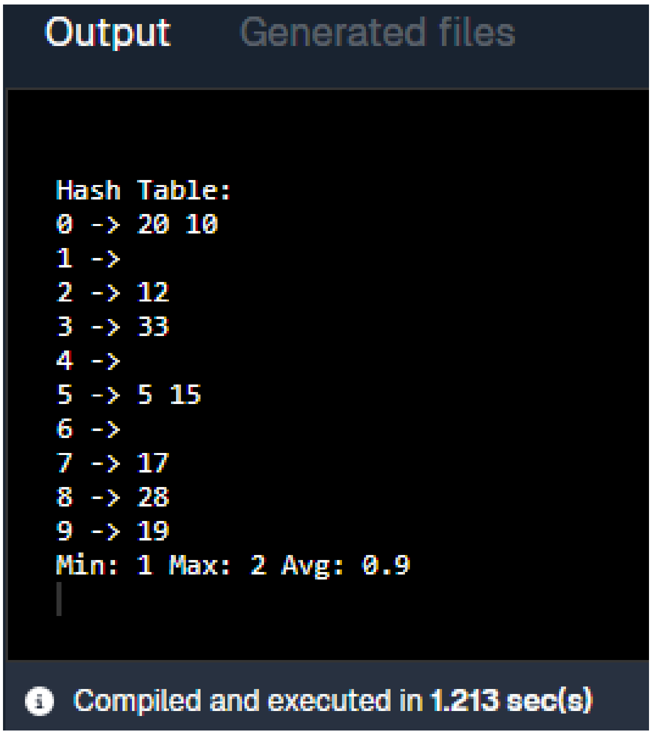
Hashing

54. Insert following keys 5, 28, 19, 15, 20, 33, 12, 17 and 10 in hash table using chaining hashing method and find minimum, maximum and average chain length in hash table.

Code:-

```
import java.util.*;
public class ChainingHashing {
    public static void main(String[] args) {
        int[] keys = {5, 28, 19, 15, 20, 33, 12, 17, 10};
        List<List<Integer>> table = new ArrayList<>();
        for (int i = 0; i < 10; i++) table.add(new ArrayList<>());
        for (int key : keys) table.get(key % 10).add(key);
        System.out.println("Hash Table:");
        for (int i = 0; i < table.size(); i++) {
            System.out.print(i + " -> ");
            for (int val : table.get(i)) System.out.print(val + " ");
            System.out.println();
        }
        int min = Integer.MAX_VALUE, max = Integer.MIN_VALUE, sum = 0;
        for (List<Integer> chain : table) {
            int size = chain.size();
            if (size > 0) {
                min = Math.min(min, size);
                max = Math.max(max, size);
                sum += size;
            }
        }
        System.out.println("Min: " + min + " Max: " + max + " Avg: " + (sum / 
10.0));
    }
}
```

Output:-



```
Hash Table:
0 -> 20 10
1 ->
2 -> 12
3 -> 33
4 ->
5 -> 5 15
6 ->
7 -> 17
8 -> 28
9 -> 19
Min: 1 Max: 2 Avg: 0.9
```

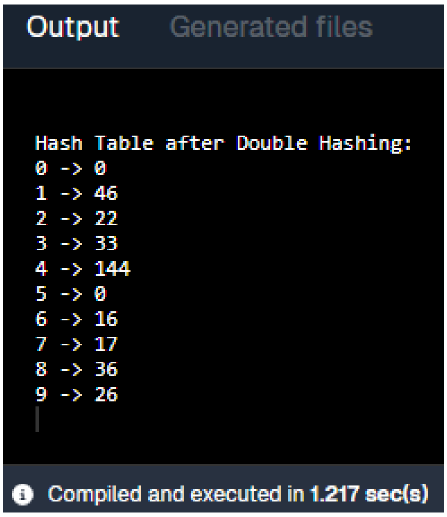
Compiled and executed in 1.213 sec(s)

55. Write a program to display hash table after inserting elements 17, 16, 22, 36, 33, 46, 26, 144 into a hash table of size 10, using Double hashing, where h(x) = x mod 10, $h_2(x)$ = x mod 6 + 1.

Code:-

```
public class DoubleHashing {
    public static void main(String[] args) {
        int[] keys = {17, 16, 22, 36, 33, 46, 26, 144};
        int[] table = new int[10];
        for (int key : keys) {
            int h1 = key % 10;
            int h2 = key % 6 + 1;
            int i = 0, idx;
            do {
                idx = (h1 + i * h2) % 10;
                i++;
            } while (table[idx] != 0);
            table[idx] = key;
        }
        System.out.println("Hash Table after Double Hashing:");
        for (int i = 0; i < table.length; i++) {
            System.out.println(i + " -> " + table[i]);
        }
    }
}
```

Output:-



```
Hash Table after Double Hashing:
0 -> 0
1 -> 46
2 -> 22
3 -> 33
4 -> 144
5 -> 0
6 -> 16
7 -> 17
8 -> 36
9 -> 26
```

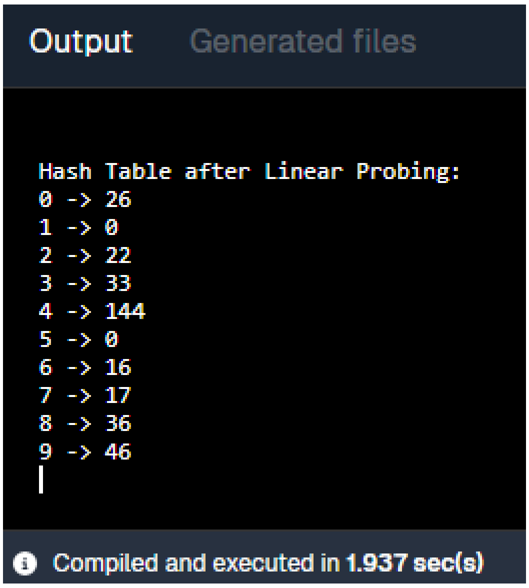
Compiled and executed in 1.217 sec(s)

56. Write a program to display hash table after inserting elements 17, 16, 22, 36, 33, 46, 26, 144 into a hash table of size 10, using linear probing, where h(x) = x mod 10.

Code:-

```
public class LinearProbing {
    public static void main(String[] args) {
        int[] keys = {17, 16, 22, 36, 33, 46, 26, 144};
        int[] table = new int[10];

        for (int key : keys) {
            int idx = key % 10;
            while (table[idx] != 0) {
                idx = (idx + 1) % 10;
            }
            table[idx] = key;
        }
        System.out.println("Hash Table after Linear Probing:");
        for (int i = 0; i < table.length; i++) {
            System.out.println(i + " -> " + table[i]);
        }
    }
}
```

Output:-



```
Hash Table after Linear Probing:
0 -> 26
1 -> 0
2 -> 22
3 -> 33
4 -> 144
5 -> 0
6 -> 16
7 -> 17
8 -> 36
9 -> 46
```

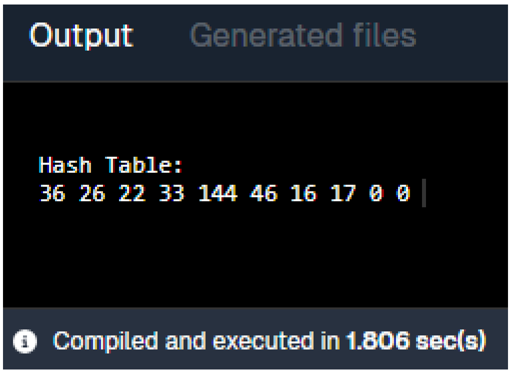
Compiled and executed in 1.937 sec(s)

57. Write a program to display hash table after inserting elements 17, 16, 22, 36, 33, 46, 26, 144 into a hash table of size 10, using quadratic probing, where h(x) = x mod 10.

Code:-

```
public class QuadraticProbing {
    public static void main(String[] args) {
        int[] keys = {17, 16, 22, 36, 33, 46, 26, 144};
        int[] table = new int[10];

        for (int key : keys) {
            int h = key % 10, i = 0, idx;
            do {
                idx = (h + i * i) % 10;
                i++;
            } while (table[idx] != 0);
            table[idx] = key;
        }
        System.out.println("Hash Table:");
        for (int n : table) System.out.print(n + " ");
    }
}
```

Output:-



```
Hash Table:
36 26 22 33 144 46 16 17 0 0
```