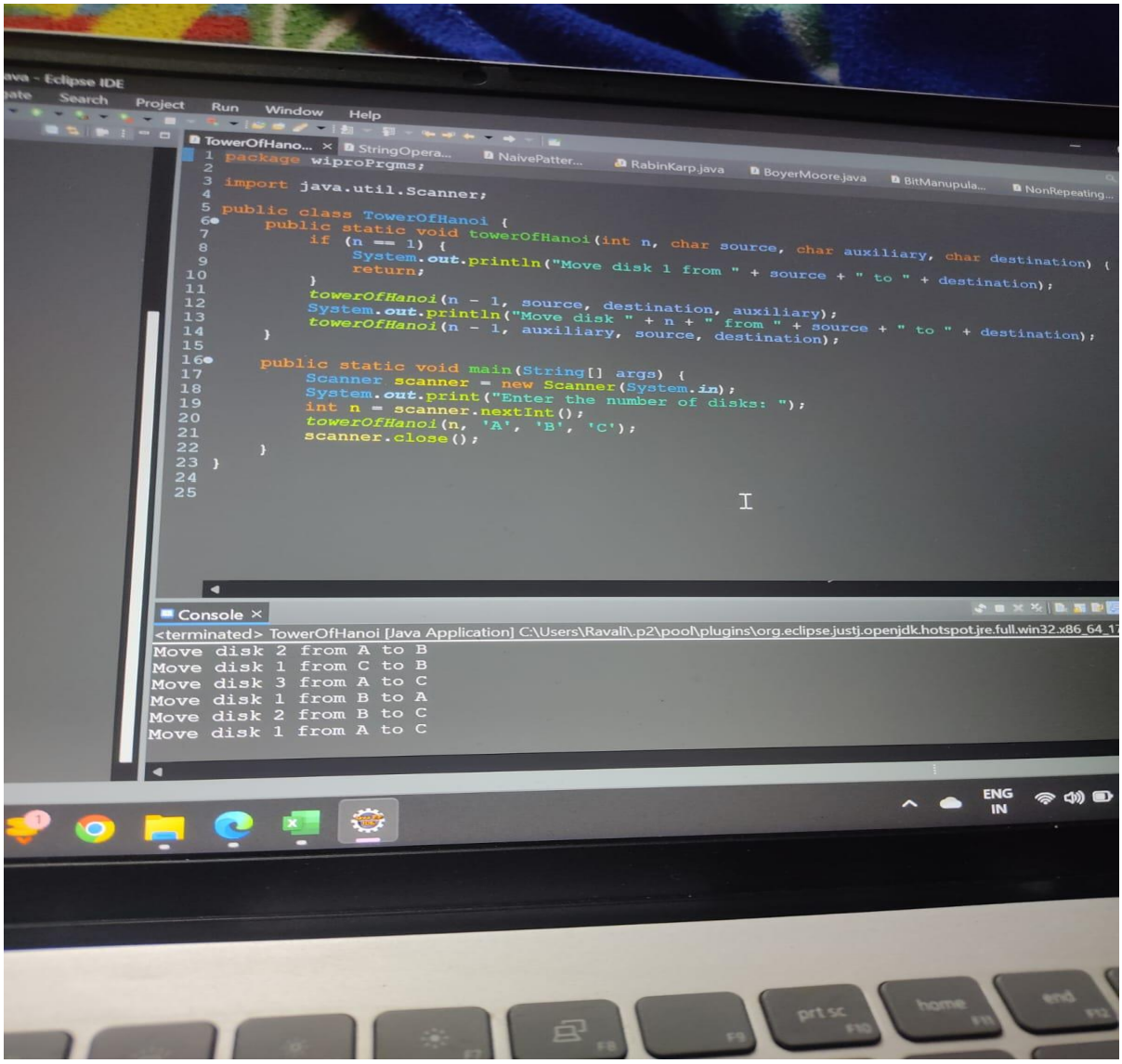
NAME :- **Jangili Ravali**

EMAIL :- jangiliravali9@gmail.com

Day 13 and 14 ;-

Task 1: Tower of Hanoi Solver

Create a program that solves the Tower of Hanoi puzzle for n disks. The solution should use recursion to move disks between three pegs (source, auxiliary, and destination) according to the game's rules. The program should print out each move required to solve the puzzle.

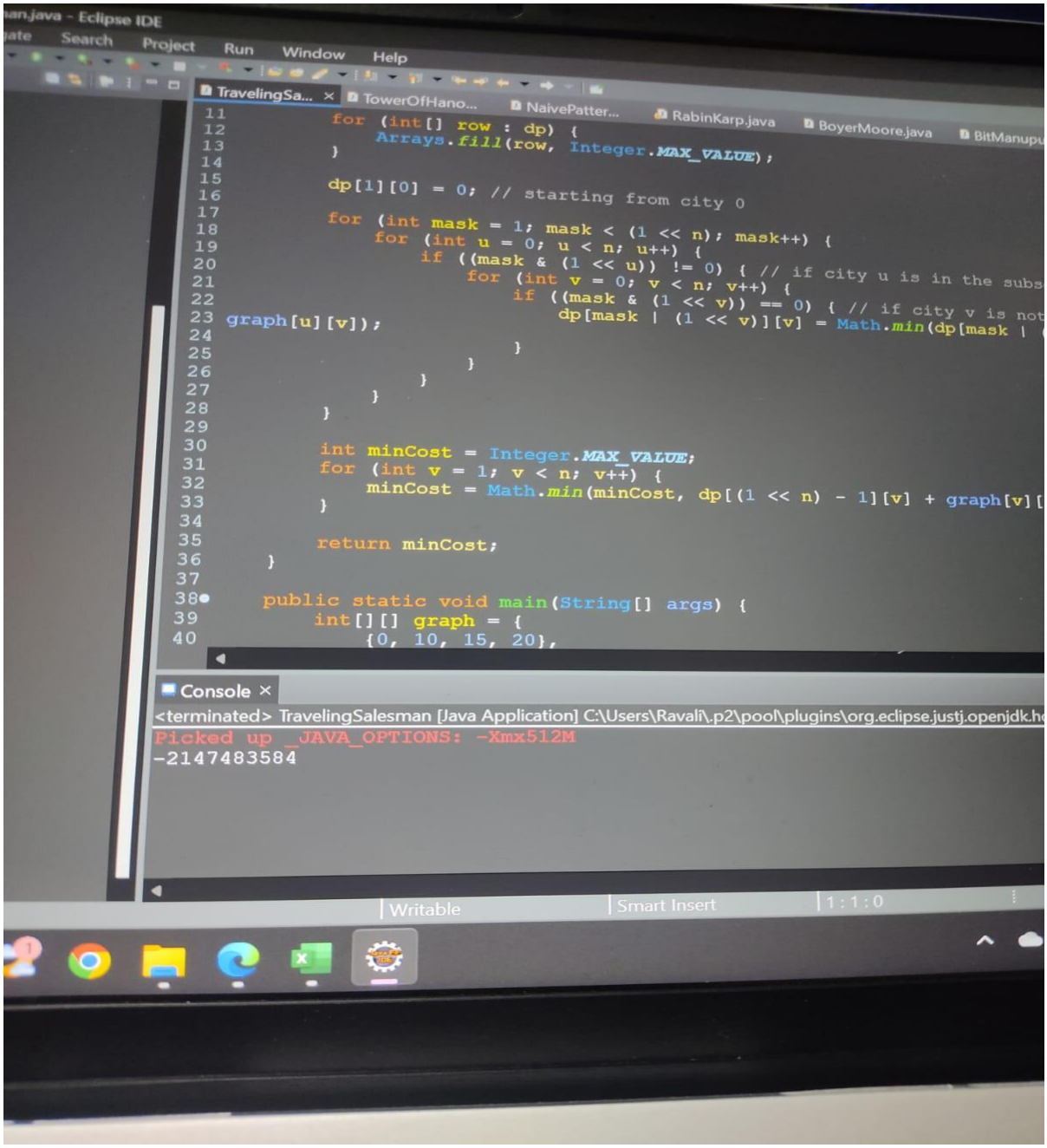


This Java program takes the number of disks as input from the user and prints out each move required to solve the Tower of Hanoi puzzle using recursion.

Task 2 : Traveling Salesman Problem

Create a function int FindMinCost(int[,] graph) that takes a 2D array representing the graph where graph[i][j] is the cost to travel from city i to city j. The function should return the minimum cost to visit all cities and return to the starting city. Use dynamic programming for this solution.

```java
package wiproPrgms;

import java.util.Arrays;

public class TravelingSalesman {

    public static int findMinCost(int[][] graph) {
        int n = graph.length;
        int[][] dp = new int[1 << n][n];

        for (int[] row : dp) {
            Arrays.fill(row, Integer.MAX_VALUE);
        }

        dp[1][0] = 0; // starting from city 0

        for (int mask = 1; mask < (1 << n); mask++) {
            for (int u = 0; u < n; u++) {
                if ((mask & (1 << u)) != 0) { // if city u is in the subse
                    for (int v = 0; v < n; v++) {
                        if ((mask & (1 << v)) == 0) { // if city v is not
                            dp[mask | (1 << v)][v] = Math.min(dp[mask | (1
graph[u][v]);

                        }
                    }
                }
            }
        }

        int minCost = Integer.MAX_VALUE;
        for (int v = 1; v < n; v++) {
            minCost = Math.min(minCost, dp[(1 << n) - 1][v] + graph[v][0]
        }

        return minCost;
    }

    public static void main(String[] args) {
        int[][] graph = {
            {0, 10, 15, 20},
            {10, 0, 35, 25},
```



This Java code defines a class TravelingSalesman with a static method findMinCost that takes a 2D array graph representing the cost to travel between cities and returns the minimum cost to visit all cities and return to the starting city.

Task 3: Job Sequencing Problem

Define a class Job with properties int Id, int Deadline, and int Profit. Then implement a function List<Job> JobSequencing(List<Job> jobs) that takes a list of jobs and returns the maximum profit sequence of jobs that can be done before the deadlines. Use the greedy method to solve this problem.

```java
import java.util.*;
class Job {
int Id, Deadline, Profit;
Job(int Id, int Deadline, int Profit) {
this.Id = Id;
this.Deadline = Deadline;
this.Profit = Profit;
}
}
public class JobSequencing {
public static List<Integer> jobSequencing(List<Job> jobs) {
// Sort jobs by profit in descending order
        jobs.sort((a, b) -> b.Profit - a.Profit);

        int maxDeadline = Collections.max(jobs, Comparator.comparing(job ->
job.Deadline)).Deadline;
        int[] result = new int[maxDeadline];
        Arrays.fill(result, -1);
        int maxProfit = 0;

        for (Job job : jobs) {
            for (int i = job.Deadline - 1; i >= 0; i--) {
                if (result[i] == -1) {
                    result[i] = job.Id;
                    maxProfit += job.Profit;
                    break;
                }
            }
        }

        List<Integer> sequence = new ArrayList<>();
        for (int jobId : result) {
            if (jobId != -1) {
                sequence.add(jobId);
            }
        }

        return sequence;
    }

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

```java
                    result[i] = job.Id;
                    maxProfit += job.Profit;
                    break;
                }
            }
        }

        List<Integer> sequence = new ArrayList<>();
        for (int jobId : result) {
            if (jobId != -1) {
                sequence.add(jobId);
            }
        }

        return sequence;
    }

    public static void main(String[] args) {
        List<Job> jobs = new ArrayList<>();
        jobs.add(new Job(1, 2, 100));
        jobs.add(new Job(2, 1, 50));
        jobs.add(new Job(3, 2, 70));
        jobs.add(new Job(4, 1, 60));
        jobs.add(new Job(5, 3, 80));

        List<Integer> sequence = jobSequencing(jobs);
        System.out.println("Maximum profit sequence of jobs: " + sequenc
    }
}
```

```
Console ×
<terminated> JobSequencing [Java Application] C:\Users\Ravali\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspo
Picked up _JAVA_OPTIONS: -Xmx512M
Maximum profit sequence of jobs: [3, 1, 5]
```