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.

```java
package Assignment13_14;

public class owerofHanoi {
    public static void main(String[] args) { hanoi( n: 3, rodFrom: "A", rodMiddle: "B", rodTo: "C");
    }
    private static void hanoi(int n, String rodFrom, String rodMiddle, String rodTo) {  3 usages
        if(n==1) {
            System.out.println("Disk 1 moved from " + rodFrom +" to " + rodTo);
            return;
        }
        hanoi( n: n-1,rodFrom, rodTo, rodMiddle);
        System.out.println("Disk " + n + " moved from " + rodFrom + " to "
                +rodTo);
        hanoi( n: n-1,rodMiddle, rodFrom, rodTo);
    }
}
```

Output:

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
Disk 1 moved from A to C
Disk 2 moved from A to B
Disk 1 moved from C to B
Disk 3 moved from A to C
Disk 1 moved from B to A
Disk 2 moved from B to C
Disk 1 moved from A to C

Process finished with exit code 0
```

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 Assignment13_14;
import java.util.Arrays;
public class TravelingSalesman {
    public static int findMinCost(int[][] graph) { int n = graph.length;  1 usage
        int[][] dp = new int[1 << n][n]; for (int[] row : dp) {
            Arrays.fill(row, Integer.MAX_VALUE);
        }
        dp[1][0] = 0;
        for (int mask = 0; mask < (1 << n); mask++) { for (int i = 0; i < n; i++) {
            if ((mask & (1 << i)) != 0) {
                for (int j = 0; j < n; j++) { if ((mask & (1 << j)) == 0) {
                    int newMask = mask | (1 << j); dp[newMask][j] = Math.min(dp[newMask][j], dp[mask][i] + graph[i][j]);
                }
                }
            }
        }
        }
        int minCost = Integer.MAX_VALUE; for (int i = 1; i < n; i++) {
            minCost = Math.min(minCost, dp[(1 << n) - 1][i] + graph[i][0]);
        }
        return minCost;
    }
```

```java
    }
    public static void main(String[] args) {
        int[][] graph = {
                {0, 10, 15, 20},
                {10, 0, 35, 25},
                {15, 35, 0, 30},
                {20, 25, 30, 0}
        };
        int minCost = findMinCost(graph);
        System.out.println("Minimum cost to visit all cities and return to the starting city: " + minCost);
    }
}
```

Output

```
Minimum cost to visit all cities and return to the starting city: 80
```

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
package Assignment13_14;
import java.util.ArrayList;
public class JobSequence {
    class Job {  10 usages
        String name; int deadline;  2 usages
        int profit; public Job(String name, int deadline, int profit) { this.name = name;  4 usages
            this.deadline = deadline; this.profit = profit;
        }
    }
    public static void main(String[] args) {
        JobSequence jobSequence = new JobSequence();
        ArrayList<Job> jobs = new ArrayList<>();
        jobs.add(jobSequence.new Job( name: "Devs",  deadline: 3,  profit: 35));
        jobs.add(jobSequence.new Job( name: "HR",  deadline: 4,  profit: 30));
        jobs.add(jobSequence.new Job( name: "Testing",  deadline: 4,  profit: 25));
        jobs.add(jobSequence.new Job( name: "Data AI",  deadline: 2,  profit: 20));
        jobs.add(jobSequence.new Job( name: "Quality Control",  deadline: 3,  profit: 15));
        jobs.add(jobSequence.new Job( name: "Call center",  deadline: 1,  profit: 12));
        int totalProfit = doJobSequence(jobs);
        System.out.println("Total profit after sequencing = " + totalProfit);
    }
    private static int doJobSequence(ArrayList<Job> jobs) {  1 usage
        jobs.sort((a, b) -> b.profit - a.profit);
        int maxDeadLine = Integer.MIN_VALUE; for (Job job : jobs) {
            maxDeadLine = Math.max(maxDeadLine, job.deadline);
        }
        boolean[] filledSlots = new boolean[maxDeadLine];
        int[] profits = new int[maxDeadLine];
        String[] results = new String[maxDeadLine];
        for (Job job : jobs) {
```

```
            for (int i = job.deadline - 1; i >= 0; i--) {
                if (!filledSlots[i]) {
                filledSlots[i] = true; results[i] = job.name; profits[i] = job.profit;
                break;
            }
            }
        }
        int totalProfit = 0;
        for (int profit : profits) { totalProfit += profit;
        }
        for (String result : results) {
            if (result != null) {
                System.out.print(result + " ");
            }
        }
        System.out.println();
        return totalProfit;
    }
}
```

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA C
Data AI Testing Devs HR
Total profit after sequencing = 110

Process finished with exit code 0
```