**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.**

```
public class TowerOfHanoi {
public static void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
    if (n == 1) {
      System.out.println("Move disk 1 from " + from_rod + " to " + to_rod);
      return;
    }
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);  rod
    System.out.println("Move disk " + n + " from " + from_rod + " to " + to_rod);
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
  }

  public static void main(String[] args) {
    int numDisks = 3;
    towerOfHanoi(numDisks, 'A', 'C', 'B');
  }
}
```

**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.**

```
public class MinCostTravel {

  public static int FindMinCost(int[][] graph) {
    int numCities = graph.length;
    int[][] dp = new int[numCities][1 << numCities];
    for (int i = 0; i < numCities; i++) {
      Arrays.fill(dp[i], Integer.MAX_VALUE);
    }
    for (int i = 0; i < numCities; i++) {
      dp[i][1 << i] = 0;
    }
    for (int mask = 1; mask < (1 << numCities); mask++) {
      for (int i = 0; i < numCities; i++) {

        if ((mask & (1 << i)) == 0) {
          for (int j = 0; j < numCities; j++) {
            if ((mask & (1 << j)) != 0) {
              dp[i][mask] = Math.min(dp[i][mask], dp[j][mask] + graph[j][i]);
            }
          }
        }
      }
    }
    int allCitiesMask = (1 << numCities) - 1;
```

```
        return dp[0][allCitiesMask] == Integer.MAX_VALUE ? -1 : dp[0][allCitiesMask];
    }

    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);
        if (minCost != -1) {
            System.out.println("The minimum cost to visit all cities and return to the starting city
is: " + minCost);
        } else {
            System.out.println("No solution exists for visiting all cities.");
        }
    }
}
```

**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.**

```
class Job {
  int id;
  int deadline;
  int profit;

  Job(int id, int deadline, int profit) {
    this.id = id;
    this.deadline = deadline;
    this.profit = profit;
  }
}

public class JobScheduling {

  public static List<Job> JobSequencing(List<Job> jobs) {
    jobs.sort((job1, job2) -> Integer.compare(job2.profit, job1.profit));
    List<Job> selectedJobs = new ArrayList<>();
    int maxDeadline = 0;
    for (Job job : jobs) {
      maxDeadline = Math.max(maxDeadline, job.deadline);
    }
    boolean[] slots = new boolean[maxDeadline + 1];
    for (Job job : jobs) {
      for (int deadline = job.deadline; deadline > 0; deadline--) {
        if (!slots[deadline]) {
          slots[deadline] = true;
          selectedJobs.add(job);
          break;
        }
      }
    }

    return selectedJobs;
  }

  public static void main(String[] args) {
    List<Job> jobs = new ArrayList<>();
    jobs.add(new Job(1, 2, 100));
    jobs.add(new Job(2, 1, 19));
    jobs.add(new Job(3, 2, 27));
    jobs.add(new Job(4, 1, 25));
    jobs.add(new Job(5, 3, 15));

    List<Job> scheduledJobs = JobSequencing(jobs);
    System.out.println("Jobs scheduled for maximum profit:");
    for (Job job : scheduledJobs) {
      System.out.println("Job ID: " + job.id + ", Deadline: " + job.deadline + ", Profit: " +
job.profit);
    }
  }
}
```