# 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.**
**Ans)**
**Code:-**

```java
package com.wipro.computalgo;
public class TowerofHanoi {

    public static void main(String[] args) {
            hanoi(3,"A","B","C");

    }
    private static void hanoi(int n, String rodFrom, String rodMiddle, String rodTo) {

            if(n==1) {
                    System.out.println("Disk 1 moved from " + rodFrom +" to " +
rodTo);

                    return;
            }

            hanoi(n-1,rodFrom, rodTo, rodMiddle);

            System.out.println("Disk " + n + " moved from " + rodFrom + " to "
+rodTo);
            hanoi(n-1,rodMiddle, rodFrom, rodTo);
    }
}
```

**OUTPUT:-**

```
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
```

**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.**

**Ans)**

**Code:-**

```java
package WiprpTask;
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;
      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;
  }
  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.**
Ans)
<u>Code:-</u>

```java
package com.wipro.patterns;
import java.util.ArrayList;
public class JobSequence {
  class Job {
    String name;
    int deadline;
    int profit;
    public Job(String name, int deadline, int profit) {
      this.name = name;
      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("Devs", 3, 35));
    jobs.add(jobSequence.new Job("HR", 4, 30));
    jobs.add(jobSequence.new Job("Testing", 4, 25));
    jobs.add(jobSequence.new Job("Data AI", 2, 20));
    jobs.add(jobSequence.new Job("Quality Control", 3, 15));
    jobs.add(jobSequence.new Job("Call center", 1, 12));
    int totalProfit = doJobSequence(jobs);
    System.out.println("Total profit after sequencing = " + totalProfit);
  }
  private static int doJobSequence(ArrayList<Job> jobs) {
    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;
```

```java
            }
        }
    }
    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;
  }
}
```

## OUTPUT:-

```
Data AI Testing Devs HR
Total profit after sequencing = 110
```