Solving an optimization problem using DP

1. Identify the problem as an optimization problem.

2. Identify the solution space having all the feasible solutions. We need to find the optimal feasible solution among all the feasible solutions. Optimality is defined by the objective function. Define the proper optimization problem in the form of

**max f(x) such that g(x) is true FOR\_ALL x BELONGS\_TO S**

3. Outline an exhaustive search (iterative) algorithm. If possible, mention the time complexity.

4. Find a way of partitioning the solution space (at least covering the solution space exhaustively). We are applying the divide-solution-space-and-conquer technique.

5. **Write the recurrence for the generalization of the given problem using the divide-solution-space-and-conquer technique.**

6. Find the base case (all edge cases) of the recurrence. Now, we have a mathematical recursive solution.

7. Write the Naive Recursive algorithm. It is a naive implementation of the recurrence with the base cases.

8. If possible, find the time complexity of the naive recursive algorithm.

9. Find the number of distinct sub-problems we have. If this number is lesser than the number of problems we are solving in the naive algorithm, we have an overlapping subproblems issue.

10. Memoization solves the overlapping sub-problems issue. Design a memoization table that can store solutions for all the conceivable sub-problems.

11. Apply memoization to the naive algorithm. Find the time and space complexity of the Memoization algorithm.

12. Find a bottom-up order of solving all the sub-problems such that whenever we are solving a sub-problem, all of its sub-problems are solved before in the bottom-up order. The bottom-up order is a topological order of the Directed Acyclic Graph (DAG) formed out of the dependencies of the sub-problems.

13. Write the DP bottom-up algorithm. Fill the table with base cases and solve all the sub-problems in a bottom-up order using the recurrence, but fetch the answers to the sub-problems from the table. Find the time and space complexity of the algorithm.

14. If the recurrence depends on sub-problems smaller by a constant, we can remember only a window of the table (instead of the entire table). We can reduce the space complexity by using either the sliding window or rolling window.