1)What are components of Dynamic Programming?

Components of Dynamic Programming

The major components in any Dynamic Programming solution are:

1. Stages
2. States and state variables
3. State Transition
4. Optimal Choice

1)Stages-When a complex problem is divided into several subproblems, each subproblem forms a stage of the solution. After calculating the solution for each stage and choosing the best ones we get to the final optimized solution.

2)States and State Variables-Each subproblem can be associated with several states. States differ in their solutions because of different choices. A state for a subproblem is therefore defined more clearly based on a parameter, which is called the state variable. It is possible in some problems, that one than one variable is needed to define a state distinctly. In these cases, there are more than one state variables.

3)State Transition-State Transition simply refers to how one subproblem relates to other subproblems. By using this state transition, we calculate our end solution.

For example, in the climbing stairs problem, the state transition was:

```
ans[n] = cost[n] + min(ans[n-1],ans[n-2])
```

4)Optimal Choice-At each stage, we need to choose the option which leads to the most desirable solution. Choosing the most desirable option at every stage will eventually lead to an optimal solution in the end

2)Differentiate dynamic programming from greedy approach

Feature	Greedy method	Dynamic programming
Feasibility	In a greedy Algorithm, we make whatever choice seems best at the moment in the hope that it will lead to global	In Dynamic Programming we make decision at each step considering current problem and solution to previously solved sub

Feature	Greedy method	Dynamic programming
	optimal solution.	problem to calculate optimal solution .
Optimality	In Greedy Method, sometimes there is no such guarantee of getting Optimal Solution.	It is guaranteed that Dynamic Programming will generate an optimal solution as it generally considers all possible cases and then choose the best.
Recursion	A greedy method follows the problem solving heuristic of making the locally optimal choice at each stage.	A Dynamic programming is an algorithmic technique which is usually based on a recurrent formula that uses some previously calculated states.
Memoization	It is more efficient in terms of memory as it never look back or revise previous choices	It requires dp table for memoization and it increases it's memory complexity.
Time complexity	Greedy methods are generally faster. For example, Dijkstra's shortest path algorithm takes O(ELogV + VLogV) time.	Dynamic Programming is generally slower. For example, Bellman Ford algorithm takes O(VE) time.
Fashion	The greedy method computes its solution by making its choices in a serial forward fashion, never looking back or revising previous choices.	Dynamic programming computes its solution bottom up or top down by synthesizing them from smaller optimal sub solutions.
Example	Fractional knapsack .	0/1 knapsack problem

3)Difference between memoization and tabulation in dynamic programming.

What is tabulation?

Tabulation is a technique that is used to implement the DP algorithms. It is also known as a bottom-up approach. It starts from solving the lowest level sub-problem. The solution to the lowest level sub-problem will help to solve next level sub-problem, and so forth. We solve all the sub-problems iteratively until we solve all the sub-problems. This approach saves the time when a sub-problem needs a solution of the sub-problem that has been solved before.

What is Memoization?

Memoization is a technique that is used to implement the DP algorithms. Memoization is also known as a top-down approach. It starts from solving the highest-level sub-problems. Initially, it solves the highest-level subproblem and then solve the next sub-problem recursively and the next. Suppose there are two sub-problems, i.e., sub-problem A and sub-problem B. When sub-problem B is called recursively, then it can use the solution of sub-problem A, which has already been used. Since A and all the sub-problems are memoized, it avoids solving the entire recursion tree generated by B and saves computation time.

Tabulation	Memoization
In tabulation, the state transition is difficult to create.	State transition relation is easy to create.
Code becomes difficult when lots of conditions are needed.	Code is not complicated and it's not difficult to create.
It is fast as the result of the previously solved sub-problems can be directly accessed from the table.	It is slow because lots of recursive calls and return statements are required.
In a tabulated version, all the entries must be filled one by one.	In a memoized version, entries in the table are filled on demand.

4)A dynamic programming solution needs problems to have an optimal substructure.Justify.

In computer science, a problem is said to have **optimal substructure** if an optimal solution can be constructed from optimal solutions of its subproblems. This property is used to determine the usefulness of dynamic programming and greedy algorithms for a problem.[1]

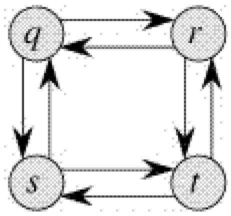
Typically, a greedy algorithm is used to solve a problem with optimal substructure if it can be proven by induction that this is optimal at each step.[1]

The “optimal substructure” property means that you can express the solution to the overall problem in terms of the solutions to subproblems. This is not really a property specific to dynamic programming, but a more general property of recursion. The essence of recursion is that you can decompose the overall problem into smaller pieces, whose solutions are then combined in some way to generate the solution to the overall problem. Dynamic programming is an optimization of recursion, so we need optimal substructure for it as well. In **dynamic programming a given problems has Optimal Substructure Property** if optimal solution of the given problem can be obtained by using optimal solutions of its sub problems.

For example the shortest path problem has following optimal substructure property: If a node X lies in the shortest path from a source node U to destination node V then the shortest path from U to V is combination of shortest path from U to X and shortest path from X to V.

But Longest path problem doesn’t have the Optimal Substructure property. i.e the longest path between two nodes doesn't have to be the longest path between the in between nodes.

For example, the longest path q->r->t is not a combination of longest path from q to r and longest path from r to t, because the longest path from q to r is q->s->t->r.



So here: **optimal solution to a problem does not contain an optimal solution to the sub problems.**