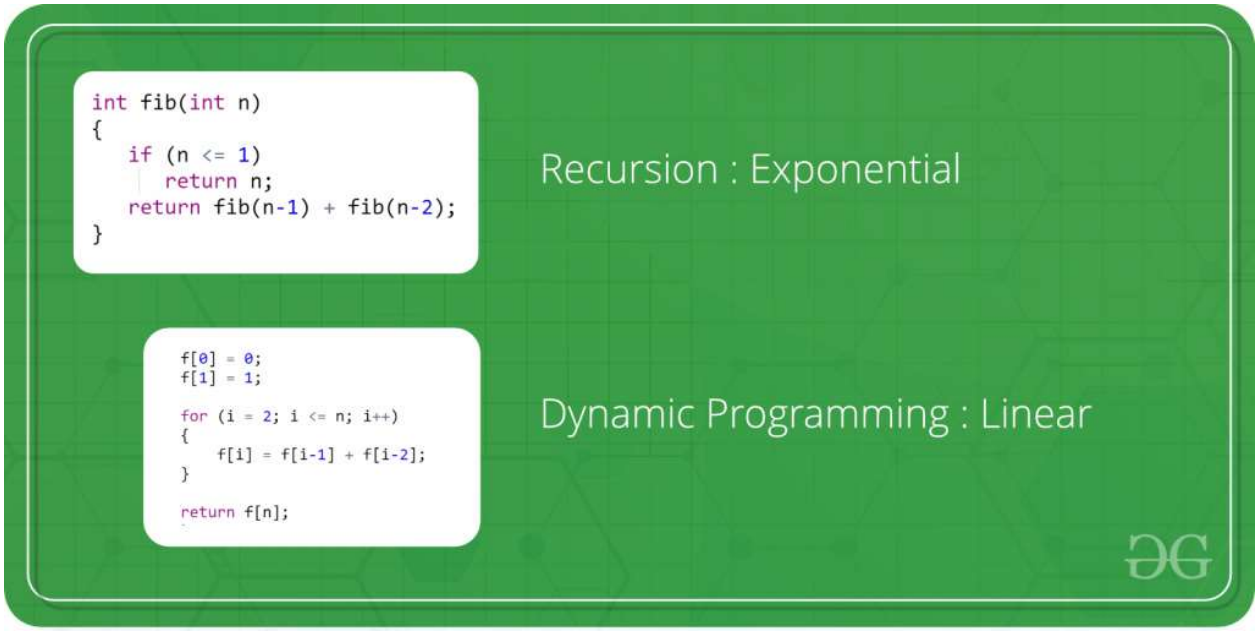
Dynamic Programming: Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for the same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial. For example, if we write a simple recursive solution for Fibonacci Numbers, we get exponential time complexity and if we optimize it by storing solutions of subproblems, time complexity reduces to linear.



The two main properties of a problem that suggest that the given problem can be solved using Dynamic programming:

1) Overlapping Subproblems

2) Optimal Substructure

1) Overlapping Subproblems:

Like Divide and Conquer, Dynamic Programming combines solutions to sub-problems. Dynamic Programming is mainly used when solutions of the same subproblems are needed again and again. In dynamic programming, computed

solutions to subproblems are stored in a table so that these don't have to be recomputed. So Dynamic Programming is not useful when there are no common (overlapping) subproblems because there is no point storing the solutions if they are not needed again. For example, Binary Search doesn't have common subproblems.

2) Optimal Substructure: A given problem has Optimal Substructure Property if optimal solution of the given problem can be obtained by using optimal solutions of its subproblems.

There are two different ways to store the values so that the values of a sub-problem can be reused. Here, will discuss two patterns of solving DP problem:

1. Tabulation: Bottom Up
2. Memoization: Top Down

	Tabulation	Memoization
State	State Transition relation is difficult to think	State transition relation is easy to think
Code	Code gets complicated when lot of conditions are required	Code is easy and less complicated
Speed	Fast, as we directly access previous states from the table	Slow due to lot of recursive calls and return statements
Subproblem solving	If all subproblems must be solved at least once, a bottom-up dynamic-programming algorithm usually outperforms a top-down memoized algorithm by a constant factor	If some subproblems in the subproblem space need not be solved at all, the memoized solution has the advantage of solving only those subproblems that are definitely required
Table Entries	In Tabulated version, starting from the first entry, all entries are filled one by one	Unlike the Tabulated version, all entries of the lookup table are not necessarily filled in Memoized version. The table is filled on demand.