Dynamic Programming:

There are two ways

1. Bottom – up approach (Tabulation) – uses iteration
2. Top – down approach (Memoization) – uses recursion

**BOTTOM – UP APPROACH:**

It starts at base cases and iterates further.

e.g., Fibonacci series: base cases for this are F(0) = 0 and F(1) = 1

With bottom-up, would use these base cases to calculate F(2)*F*(2), and then use that result to calculate F(3)*F*(3), and so on all the way up to F(n)*F*(*n*).

// Pseudocode for this :

F = array of length (n + 1)

F[0] = 0

F[1] = 1

For i from 2 to n :

F[i] = F[i – 1] + F[i – 2]

**bottom-up uses iteration.**

**TOP- DOWN APPROACH :**

It is implemented with recursion and made efficient with memoization.

Memoization is storing the result usually is hashmap or array and reuse the same result when the same function call is made again.

// Pseudocode for this :

memo = HashMap / array

Function F (integer i ) :

If i is 0 or 1 :

return i

if I doesn’t exist in memo :

memo[ i ] = F( i – 1 ) + F ( i – 2 )

return memo[ i ]

**Typically, top-down is implemented with a recursive function and hash map,**

**bottom-up is implemented with nested for loops and an array.**

**WHEN TO USE DP** ?

**The first characteristic** that is common in DP problems is that the problem will ask for the optimum value (maximum or minimum) of something, or the number of ways there are to do something. For example:

* What is the minimum cost of doing...
* What is the maximum profit from...
* How many ways are there to do...
* What is the longest possible...
* Is it possible to reach a certain point...

**The second characteristic** that is common in DP problems is that future "decisions" depend on earlier decisions. Deciding to do something at one step may affect the ability to do something in a later step. This characteristic is what makes a greedy algorithm invalid for a DP problem - we need to factor in results from previous decisions. Admittedly, this characteristic is not as well defined as the first one, and the best way to identify it is to go through some examples.

If you can think of an example where earlier decisions affect future decisions, then DP is applicable.

[Longest Increasing Subsequence](https://leetcode.com/problems/longest-increasing-subsequence/)

[House Robber](https://leetcode.com/problems/house-robber/)

To summarize: if a problem is asking for the maximum/minimum/longest/shortest of something, the number of ways to do something, or if it is possible to reach a certain point, it is probably greedy or DP. With time and practice, it will become easier to identify which is the better approach for a given problem. Although, in general, if the problem has constraints that cause decisions to affect other decisions, such as using one element prevents the usage of other elements, then we should consider using dynamic programming to solve the problem. **These two characteristics can be used to identify if a problem should be solved with DP.**

* Dynamic programming is used for problems with: **Optimal substructure, Overlapping subproblems**
* Usually, bottom-up algorithms are faster than top-down ones.
* Without memoization, a recursive solution for Fibonacci Number would have the time complexity: O(2ˆn)