<https://stackoverflow.com/questions/13538459/difference-between-divide-and-conquer-algo-and-dynamic-programming>

# Difference between Divide and Conquer Algo and Dynamic Programming

Q: What is the difference between Divide and Conquer Algorithms and Dynamic Programming Algorithms? How are the two terms different? I do not understand the difference between them.

Please take a simple example to explain any difference between the two and on what ground they seem to be similar.

A:

## Dynamic Programming and Divide-and-Conquer Similarities

As I see it for now I can say that **dynamic programming is an extension of divide and conquer paradigm**.

I would not treat them as something completely different. Because **they both work by recursively breaking down a problem into two or more sub-problems** of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

So why do we still have different paradigm names then and why I called dynamic programming an extension. It is because dynamic programming approach may be applied to the problem **only if the problem has certain restrictions or prerequisites**. And after that dynamic programming extends divide and conquer approach with **memoization** or **tabulation** technique.

Let’s go step by step…

## Dynamic Programming Prerequisites/Restrictions

As we’ve just discovered there are two key attributes that divide and conquer problem must have in order for dynamic programming to be applicable:

* **Optimal substructure** — optimal solution can be constructed from optimal solutions of its subproblems
* **Overlapping sub-problems** — problem can be broken down into subproblems which are reused several times or a recursive algorithm for the problem solves the same subproblem over and over rather than always generating new subproblems

Once these two conditions are met we can say that this divide and conquer problem may be solved using dynamic programming approach.

## Dynamic Programming Extension for Divide and Conquer

Dynamic programming approach extends divide and conquer approach with two techniques (**memoization** and **tabulation**) that both have a purpose of storing and re-using sub-problems solutions that may drastically improve performance. For example naive recursive implementation of Fibonacci function has time complexity of O(2^n) where DP solution doing the same with only O(n) time.

**Memoization (top-down cache filling)** refers to the technique of caching and reusing previously computed results. The memoized fib function would thus look like this:

memFib(n) {

if (mem[n] is undefined)

if (n < 2) result = n

else result = memFib(n-2) + memFib(n-1)

mem[n] = result

return mem[n]

}

**Tabulation (bottom-up cache filling)** is similar but focuses on filling the entries of the cache. Computing the values in the cache is easiest done iteratively. The tabulation version of fib would look like this:

tabFib(n) {

mem[0] = 0

mem[1] = 1

for i = 2...n

mem[i] = mem[i-2] + mem[i-1]

return mem[n]

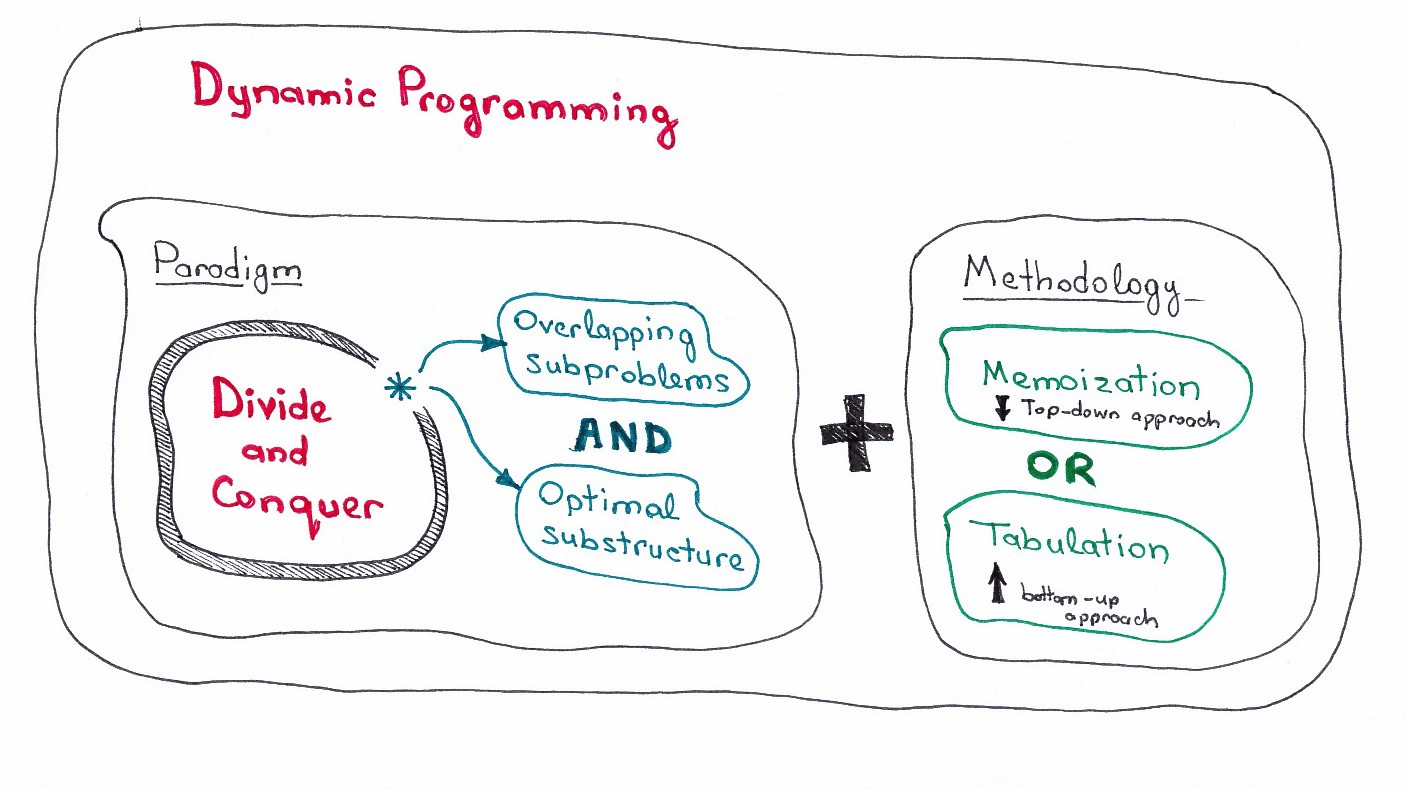
}

You may read more about memoization and tabulation comparison [here](https://programming.guide/dynamic-programming-vs-memoization-vs-tabulation.html).

The main idea you should grasp here is that because our divide and conquer problem has overlapping sub-problems the caching of sub-problem solutions becomes possible and thus memoization/tabulation step up onto the scene.

## So What the Difference Between DP and DC After All

Since we’re now familiar with DP prerequisites and its methodologies we’re ready to put all that was mentioned above into one picture.



If you want to see code examples you may take a look at [more detailed explanation here](https://trekhleb.dev/blog/2018/dynamic-programming-vs-divide-and-conquer/) where you'll find two algorithm examples: Binary Search and Minimum Edit Distance (Levenshtein Distance) that are illustrating the difference between DP and DC.