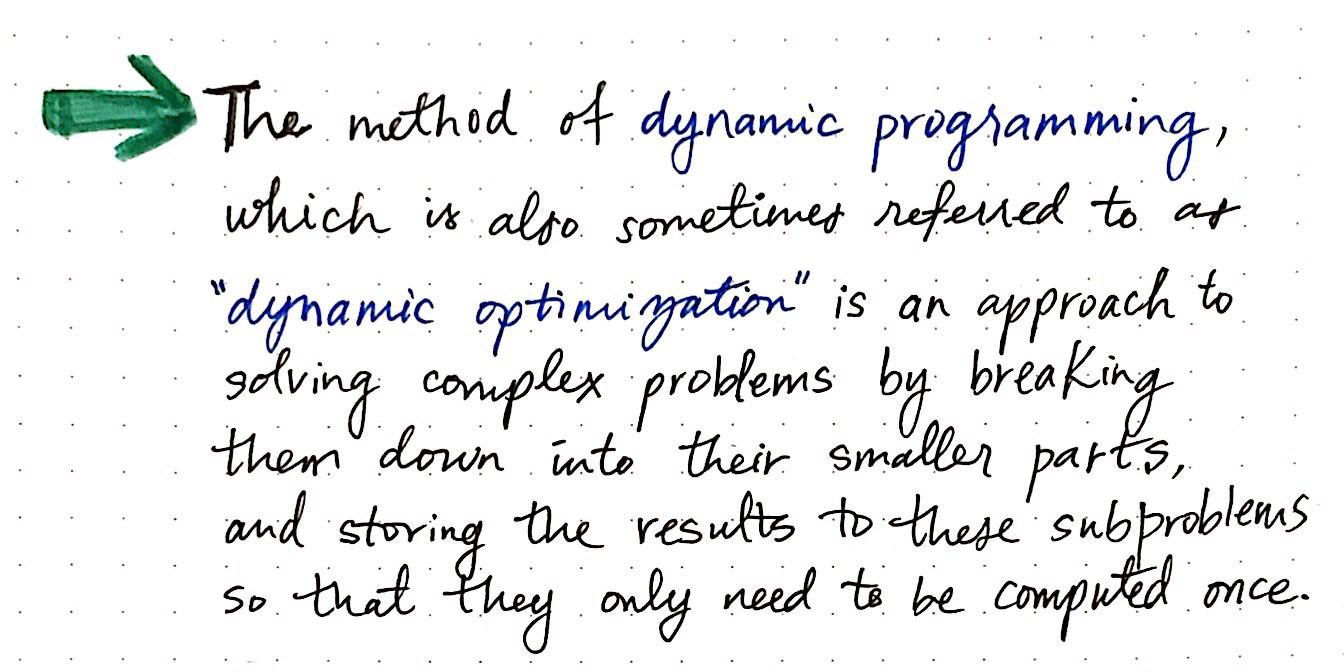


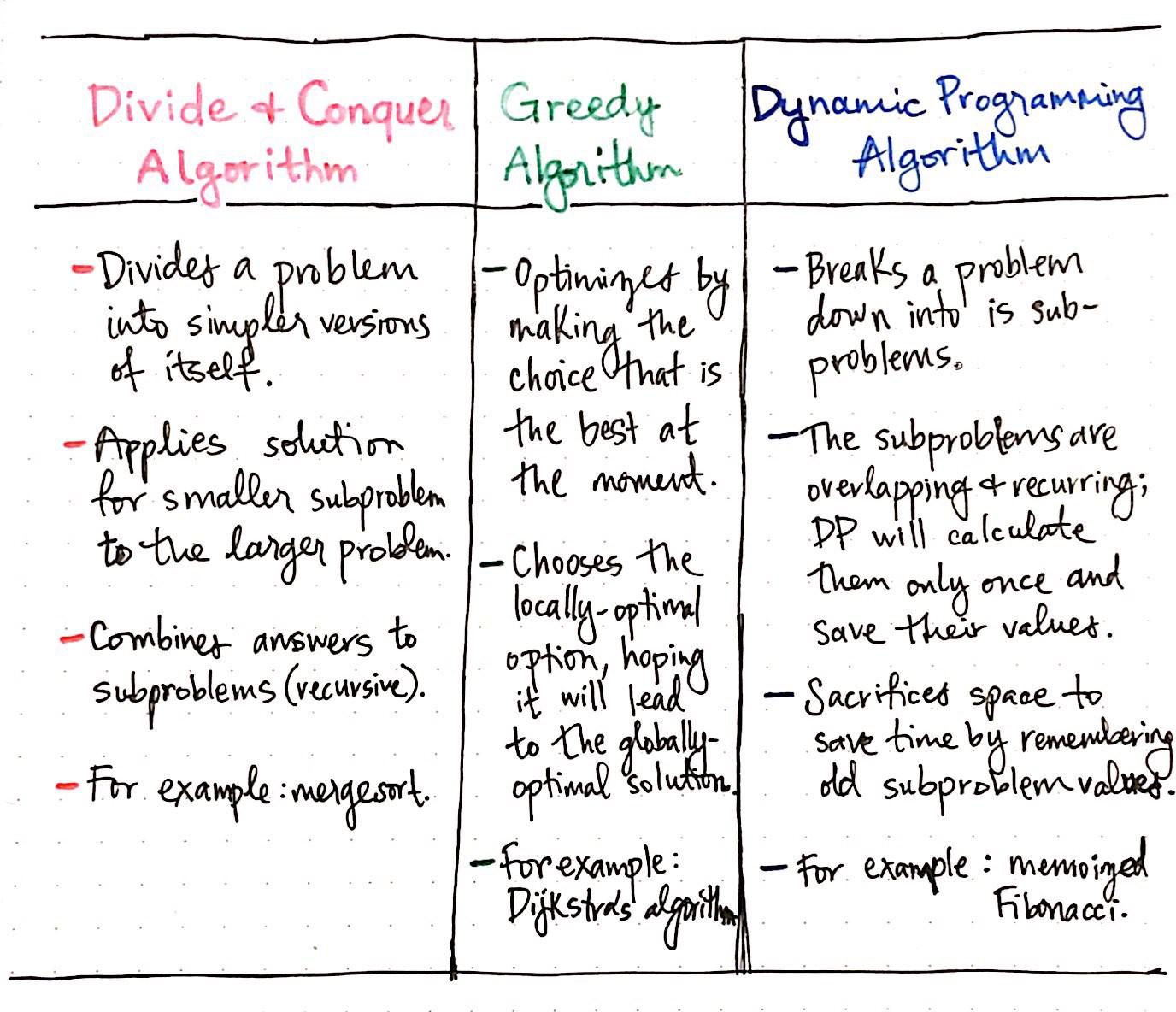
The term dynamic programming (abbreviated as “DP”)has a reputation for sounding for more intimidating than what it actuallyis. I think a large part of this has to do with its name, which — to me at least — *sounds* really complicated. But we’ll come back to the name a bit later. Let’s start with a definition in order to try to wrap our heads around what on earth “dynamic programming” means.



There are two approaches to designing algorithms that we’ve already seen in this series. One of them we know by name: the ***divide and conquer algorithm***. We’ll recall that a divide & conquer algorithm divides a problem into simpler versions of itself, and then applies the same solution that it use on the smaller, subproblems to the larger problem itself.

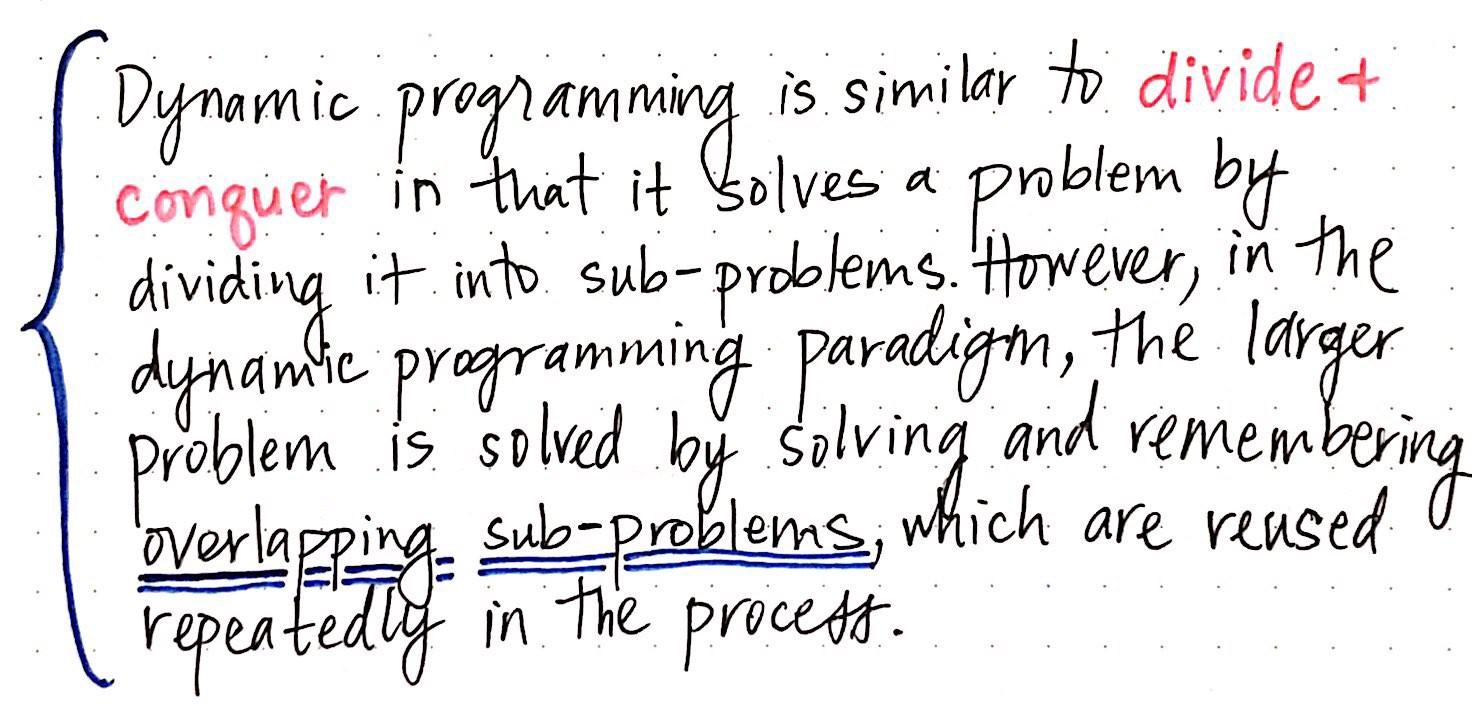
But our knowledge of algorithm design doesn’t just end with the divide and conquer approach. Greedy algorithms make the optimal choice on a local level — that is to say, the make the most efficient choice at each stage, optimistically hoping that, if they choose the best option at each point, eventually, they’ll arrive at the “global optimal choice”.

Greedy algorithms make the best choice in the moment, and then solve whatever subproblems arise from the choice that they made. One consequence of making “greedy choices” as an algorithm is that the algorithm chooses the best option it can, and never goes back to reconsider its choice. Ultimately, this means that a greedy algorithm can very well find a solution that is actually not the most optimal choice!

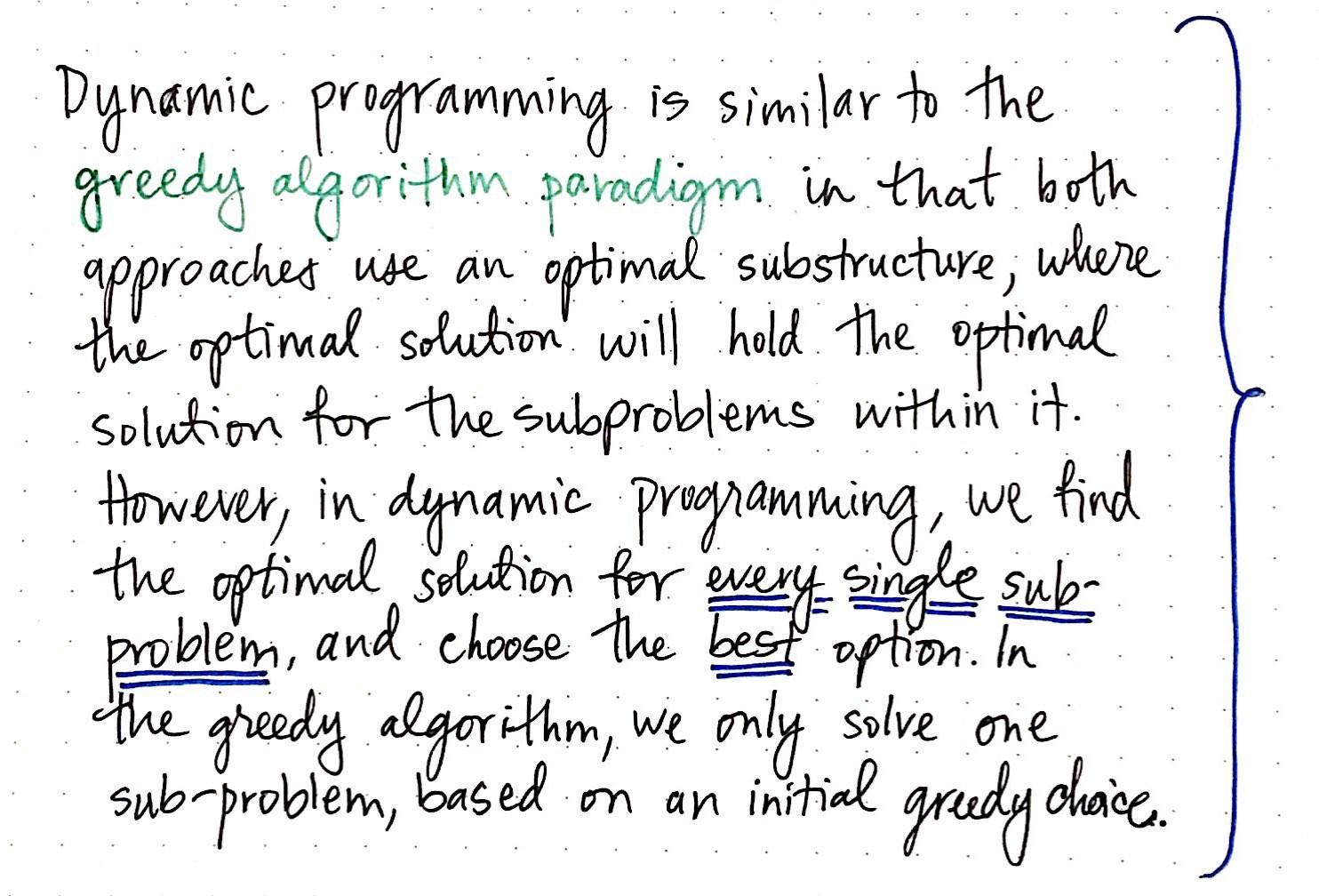


Okay, so we have *divide and conquer* algorithms, and we have *greedy* algorithms. But, let’s get back to the problem at hand: understanding how *dynamic programming* algorithms compare to these two!

Looking at the table illustrated above, we can see that DP algorithms are similar to divide and conquer in one obvious way: they both involve breaking down a large problem into smaller, simpler subproblems.



However, we’ll see one major difference between the dynamic programming approach and the divide and conquer one, too. In the DP paradigm, when we break down the larger problem into its smaller parts, those subproblems actually *overlap.*



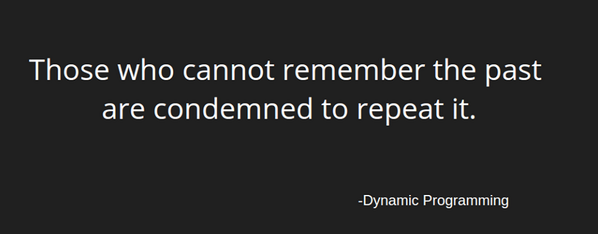
# Remember to Remember

*The trick to calculating all of the solutions to the various subproblems (and then choosing the best one) is remembering previous solutions.*

**Types**

**Top-down approach**: start with the original problem(F(n) in this case), and recursively solving smaller and smaller cases(F(i)) until we have all the ingredient to the original problem.

**Bottom-up approach**: start with the basic cases(F(1) and F(2) in this case), and solving larger and larger cases.



Top Down Vs Bottom Up

**Version 1**: To Master Dynamic Programming, I would have to practice Dynamic problems and to practice problems – Firstly, I would have to study some theory of Dynamic Programming from GeeksforGeeks

**Version 2**: I will study the theory of Dynamic Programming from GeeksforGeeks, then I will practice some problems on classic DP and hence I will master Dynamic Programming.