1 Divide and Conquer

We already saw in the *divide and conquer* paradigm how we can divide the problem into subproblems, recursively solve those, and combine those solutions to get the answer of the original problem.

Some examples of the divide and conquer paradigm are mergesort and binary search. When the subproblems are independent apart from needing to merge them together at the end, the problem may often lend itself to parallelization in the implementation.

Here's a somewhat different example:

Exercise. *Suppose you're given n points in the coordinate plane, and for convenience assume that no two of their x or y coordinates are the same. Design an algorithm that finds the shortest possible distance between a pair of these points. What is the time complexity of your algorithm? (You should assume that calculating the distance can be done in unit time.)*

2 Dynamic Programming

But for some problems, divide and conquer ends up having exponential time complexity because we will be recomputing the answer of same subproblems over and over again. To avoid recalculations, we use a lookup table. And to make it more effectively, we build the lookup table in a bottom-up fashion, until we reach the original problem.

Usually, any problem where you can write a recursion for the solution (not just for the runtime) is a good dynamic programming candidate.

Here's a good template (followed up by a runtime and/or space analysis) for writing solutions to dynamic programming problems:

1. Define a set of subproblems that can lead to the answer of the original problem.
2. Write the recursion solution of the original problem from the solution of subproblems.
3. Build the solution lookup table in a bottom-up fashion, until reaching the original problem.

- Initialize the lookup table.
- Fill the other terms by recursion.

Some things to keep in mind:

- To figure out how to write down the recursion, it's often helpful to think about how to divide into subproblems–along what axis are your subproblems "smaller" than the original?
- In many cases, the size of your lookup table is greater than the actual space complexity, since it may be enough to keep a row or two of the table in memory rather than the entire thing.

2.1 Making change with coins

Exercise. *Given a monetary system with M coins of different values $c_1, c_2, \ldots, c_M$, devise an algorithm that can make change of amount N using the minimum number of coins. What is the complexity of your algorithm?*

Exercise (Extension)**.** *Change your algorithm to compute the number of different ways of making changes for amount N with the given coins (where the order of the coins doesn't matter).*

2.2 Robot on a grid

Exercise (Basic problem)**.** *Imagine a robot sitting on the upper-left corner of an $M \times N$ grid. The robot can only move in two directions at each step: right or down. Write a program to compute the number of possible paths of the robot to the lower-right corner. What is the complexity of your program? (You can assume that integer multiplication can be done in constant time!)*

Exercise (Mathematician's solution)**.** *Can you derive a mathematical formula to directly find the number of possible paths? What is the complexity for a computer program that computes this formula?*

Exercise (Extension)**.** *Imagine that certain squares on the grid are occupied by some obstacles (probably your fellow robots, but they don't move), change your program to find the number of possible paths to get the lower-right corner without going through any of those occupied squares.*

2.3 Balanced partition

Exercise (A wedge: positive integer subset sum problem)**.** *Given a set of M positive integers $A = \{a_1, a_2, \ldots, a_M\}$, and a number N, design a program to find out whether it is possible to find a subset of A such that the sum of elements in this subset is N.*
Example: $A = \{1, 4, 12, 20, 9\}$ *and* $N = 14$*, we can find a subset* $\{1, 4, 9\}$ *such that* $1 + 4 + 9 = 14$.

Exercise (Balanced partition problem)**.** *Given a set of M positive integers $A = \{a_1, a_2, \dots, a_M\}$, find a partition A_1, A_2, such that $|S_1 - S_2|$ is minimized, where S_i is the sum of elements in subset A_i. (A partition of A means two subset A_1, A_2 such that $A = A_1 \bigcup A_2$ and $A_1 \bigcap A_2 = \emptyset$.)*
Example: $A = \{1, 4, 12, 20, 9\}$. The best balanced partition we can find is $A_1 = \{1, 12, 9\}, S_1 = 1+12+9 = 22$ and $A_2 = \{4, 20\}, S_2 = 4 + 20 = 24$.

2.4 Palindrome

A palindrome is a word (or a sequence of numbers) that can be read the same say in either direction, for example "abaccaba" is a palindrome.

Exercise (Palindrome)**.** *Design an algorithm to compute what is the minimum number of characters you need to remove from a given string to get a palindrome.*
Example: you need to remove at least 2 characters of string "abbaccdaba" to get the palindrome "abaccaba".