Lecture 3: Recursion

Recursion: it's when a function calls itself.

- Sometimes you can have functions that call each other as well and form a cycle. That's also recursion

The long answer: it's a problem solving technique that takes advantage of the fact that to solve a problem, you need to solve a smaller problem the same way.

Recursion is optional, but often elegant.

Examples: Many examples involve more complex data structures that we'll get to later in the semester, but let's start with mathematical examples:

- Implementing factorial
- Implementing a power (using a^(n-1) to calculate a^n)
    - A better way to implement power (using a^(n/2) to calculate a^n)

What do you look for when you recursion?

- The function can't ALWAYS call itself, otherwise it would add to the stack forever
- So you need a base case. That's the smallest version of the problem that you can solve.
    - Usually you look for n=0, or some object being null
- Then, in your recursive case, you need to change the parameter, the parameter needs to be SMALLER, or CLOSER TO THE BASE CASE in the new call.
    - So in our example, you have a number n decreasing by 1. If it's positive, it'll eventually get to 0.
    - If you're following pointers, and you know that eventually you'll get to a null pointer or some object that's a base case, then good!
        - Not guaranteed if you have a cycle of references.
- Then, you need to make sure that you're solving a problem which is self-similar. The answer can be built up from the answer to a smaller version of the problem
    - Example: Tower of Hanoi
        - Strategy: Solve for (n-1), then choose the same moves, but move that tower to the second row, continue

Mathematical Recurrences can be solved with recursion.

- A recurrence is a sequence whose members are defined by earlier values of the sequence. There are also boundary conditions, which are equivalent to our base cases.
- Example: Linear function of the previous value: a_n = m*a_(n-1) + b
- Example: Fibonacci: a_n = a_(n-1) + a_(n-2)
- Example: Pascal's Triangle (2d)

- All 3 of these examples have CLOSED FORMS - that means there's a formula you can use to calculate the values of the sequence outright without looking at previous values.. Sometimes it's possible to find those closed forms.
    - A lot of them found in the 19th and early 20th century where computation cost savings matters, also of interest in pure mathematics

Dynamic Programming

This is especially useful for recurrences.

- Instead of starting with the full problem and working your way down, you start from the solution on the bottom and work your way up
- This is usually done with a loop instead of recursion.
- Benefits for the Fibonacci Sequence and Pascal's Triangle - not repeating calculations.
    - Draw a diagram of all the computation and recursive calls that recursive fibonacci is making.
    - With pascal's triangle, you'd have to build up successively larger arrays
- Benefits for the linear functions are less, but without recursion there's no need to use the function stack.

In homework 0, you're going to measure the running time of the fibonacci sequence using recursion, dynamic programming, and the closed form.