Recursion

> *You jack into cybespace to see how your program is running against corp security. It is beautiful, one single orgin branched out into millions of sub functions. You watch as each targets the very base of their intrusion countermeasures... With this much help, it won't be long until we get to their Magnum Opus source code.*

What is Recursion

Recursion is when a function calls itself. When we call these methods, we give it simplier parameters, funneling it down to a simple, function that would return to it's originating function.

Example: Fibbonaci

For example, we can do recursion in math. A fibbonaci sequence a sequence that starts at 0 and 1, and every number after that is the last 2 numbers added together, like so

0 1 1 2 3 5 8 13

If we made this into a tree, we can see that a fibonacci program looks like this:

```
          13
         /  \
        8    5
       / \  / \
      5   3 3  2
... And so on...
```

Complete the tree. What do you get as the bases? **You get 1's and 0's.** This means that we can break up the function into recursion, and have our program **divide and conquer**. If we tried writing a program without recursion, it would be a formiddable program. Not impossible, but quite a workout for us. However, using recursion, we can do the following:

```
1  //This will return the value of the fibonacci sequence at n.
2  public int fibonacci(n) {
3    if(n < 3) return n - 1;
4    else return fibonacci(n-1) + fibonacci(n-2);
5  } //End fibonacci()
```

If we take the time to analyze this, we can see that the program does exactly what the tree does: **break the program up until it gets to its base values: 1 and 0**. After this, add all the 1's and 0's together for our grand total.

Stacks in Recursion

Notice how **we're calling functions within functions**. This means that we are pushing more functions onto our stack. This is why we have to be weary: **Too much function calls or a non-terminating recursion function** will cause a **Stack overflow**, mearning we do not have memory to push on another function onto the stack. Common mistakes include:

1. A never ending recursion: there is no base case that will terminate the end of a string.
2. A recursion that does not actually split the problem into smaller cases
3. The call is simply just too big