Recursion

Background: mathematical induction.

"Traditional" proofs look like this:

Goal: $A \Rightarrow E$.

$A \Rightarrow B$ (Axiom...)

$B \Rightarrow C$ (Algebra...)

$C \Rightarrow D$ (Lemma ...)

$D \Rightarrow E$ (More algebra) ✓

For inductive proofs, we are usually trying to prove a <u>parameterized</u> statement. Call it $T(n)$.

E.g. $T(n) = \left(\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \right)$

We want to show $T(n)$ holds <u>for all $n > 0$</u>.

Big picture: there are 2 main components:

① "Explicitly" prove $T(n)$ for a small value, e.g. $n = 1$.

E.g. $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$ ✓

② Show that $T(n) \Rightarrow T(n+1)$.

For the example above, it might go like this:

Assume $T(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

Then $\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1)$

$$= \frac{n(n+1)}{2} + n+1$$

$$= \frac{n(n+1)}{2} + \frac{2(n+1)}{2}$$

$$= \frac{n(n+1)+2(n+1)}{2} = \frac{(n+1)(n+2)}{2} \checkmark$$

$\therefore T(n+1)$ also true!

Conclusion: $T(n)$ is true <u>for all $n>0$</u>:

We've shown that a "traditional" proof always exists:
E.g. for $n=4$, Then we have the following proof:

$$\underset{1}{T(1)} \underset{2}{\Longrightarrow} T(2) \underset{2}{\Longrightarrow} T(3) \underset{2}{\Longrightarrow} T(4) \checkmark$$

<u>Recursion</u> It's basically (exactly?) the above,
but with $T(n)$ = "my program works on any input of size n".

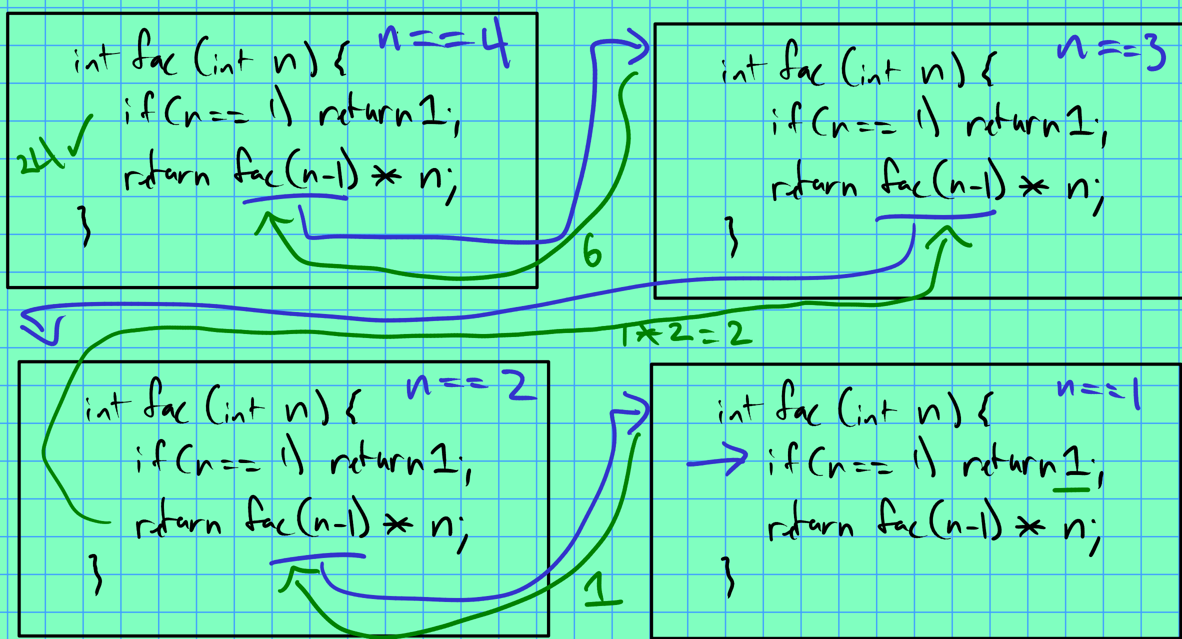
Example: computing $n! = \prod_{i=1}^{n} i \quad (= 1\cdot 2\cdot 3 \cdots n)$

```
int fac(int n)
{    if(n==1) return 1;  // 1
     // Now assume fac works on any input ≨ n ...
     // Note: n! = (n-1)! · n
     return fac(n-1) * n;  // 2
```

}

This program could be seen as an inductive proof of its own correctness!

Let's "trace" fac(4) :



Example 2: Fibonacci : $\{a_n\}_{n=0}^{\infty}$ $a_0 = a_1 = 1.$

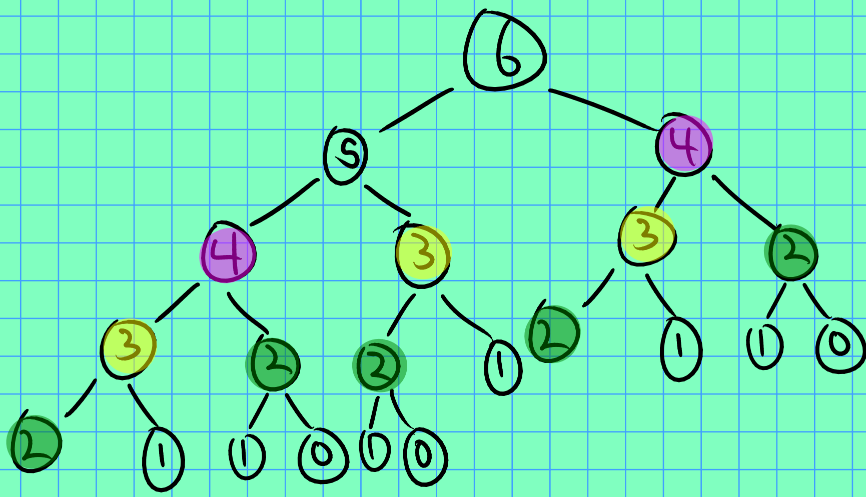
$a_n = a_{n-1} + a_{n-2}$.

1 1 2 3 5 8 11 . . .

```
int fib(int n)
{  if(n < 2) return 1;
   return fib(n-1) + fib(n-2);
}
```

// works! but on input ≥ 50, your computer gets warm... o_O

"Trace" a call to f(6):



Exercise: try to quantify the badness:

count # of total function calls that result on input n.