More on recursion

$f_n = f_{n-1} + f_{n-2}$

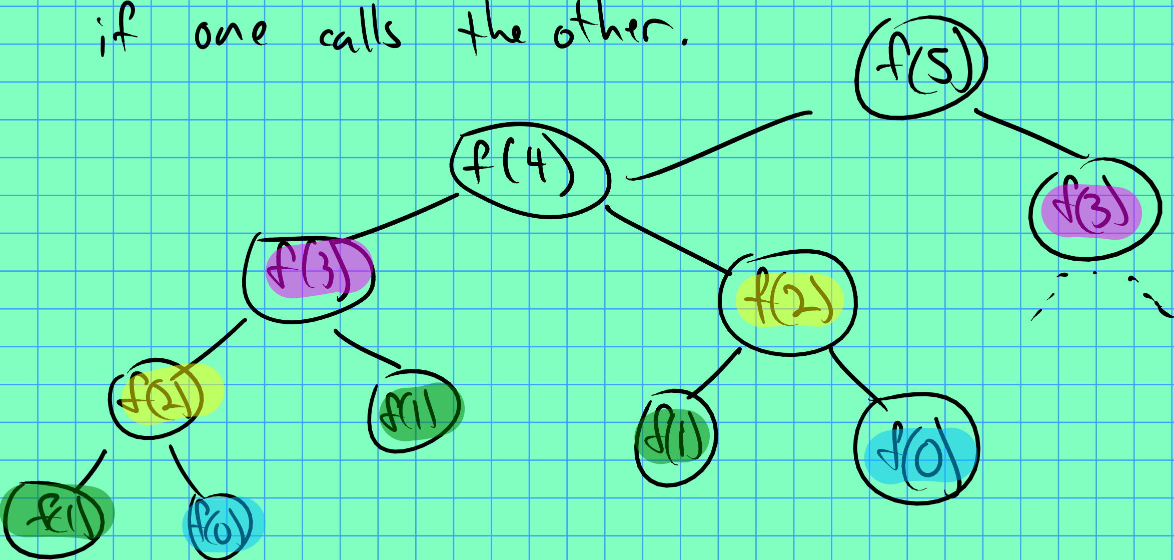
f_i	1	1	2	3	5	8	...
i	0	1	2	3	4	5	...

Fibonacci :

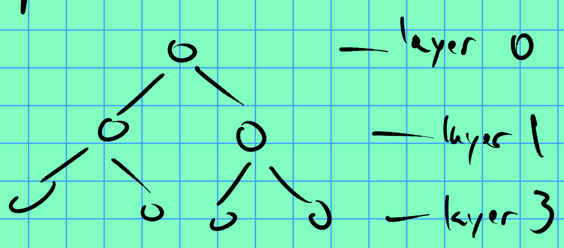
```
int fib(int n)
{   if(n < 2) return 1;
    return fib(n-1) + fib(n-2);
}
```

why so slow?!?!?

Draw recursion tree: each node corresponds to a function call. Nodes are connected if one calls the other.



Ballpark estimate for # recursive calls:



for an n-layer, complete tree, # nodes is $= 2^n - 1$.

size layer 0 = 1

size layer 1 = 2

⋮

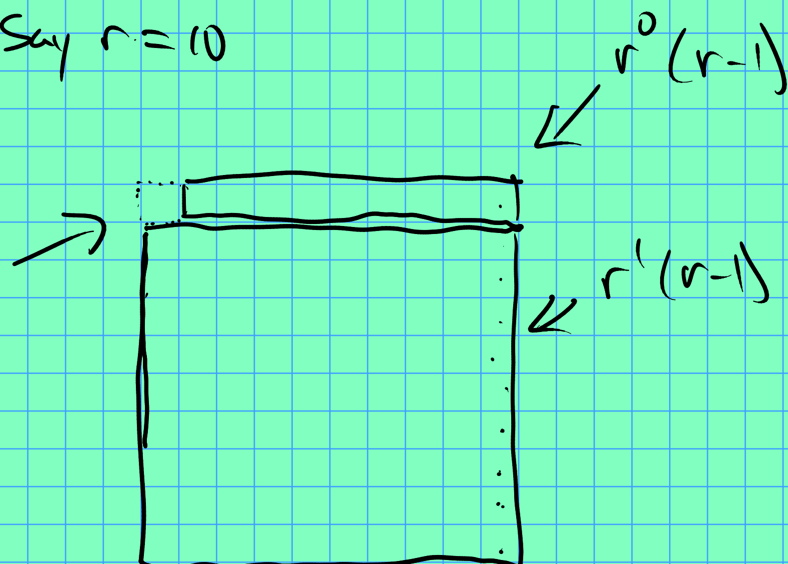
size layer i = 2^i

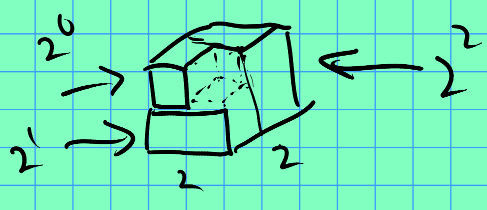
size of entire tree: $\sum_{i=0}^{n-1} 2^i = \frac{1-2^n}{1-2} = 2^n - 1$

(In general, $\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r}$.

Proof: Geometric proof: look instead at $(r-1)\sum_{i=0}^{n} r^i = r^{n+1} - 1$

Say r = 10





Algebra proof:

$$\sum_{i=0}^{n} r^i = \frac{(1-r)(1+r+r^2+r^3+\cdots+r^n)}{(1-r)}$$

$$(1-r)(1+r+r^2+r^3+\cdots+r^n)$$

$$= (1-\cancel{r}) + (\cancel{r}-\cancel{r^2}) + (\cancel{r^2}-\cancel{r^3}) + \cdots + (\cancel{r^n}-r^{n+1})$$

$$= 1 - r^{n+1}$$

Can't we have it all? Nice aesthetic + not super slow?

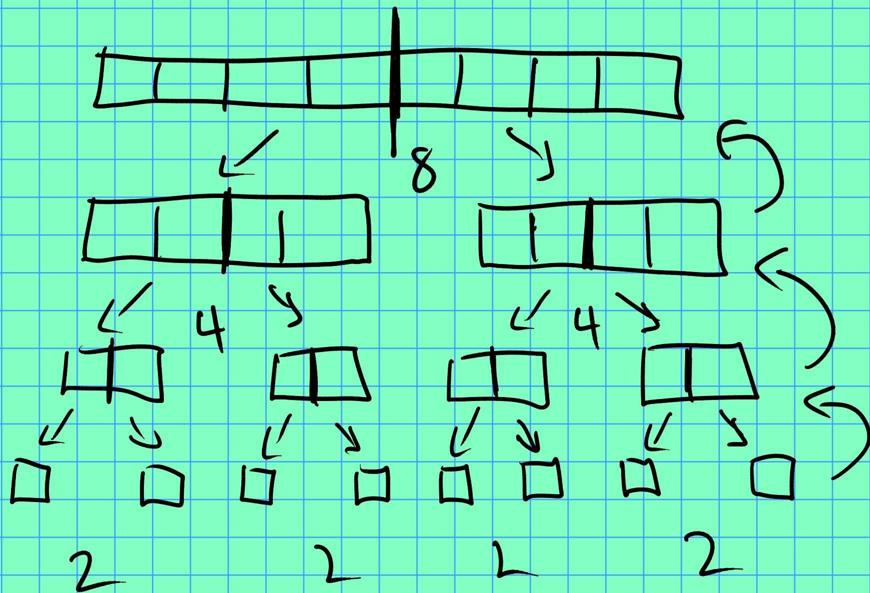
Kind of. One technique: <u>memoization</u>.

Make your function not so forgetful.

```
int fibM(int n, map<int,int> A)
{
    if(n<2) return 1;
    //Before making recursive calls,
    // check the map!
    if(A.find(n) != A.end()) { // found it!
        return A[n];
    }
    // didn't find it, so compute + save for later.
    int f = fib(n-1) + fib(n-2);
    A[n] = f;
    return f;
}
```

Let's revisit <u>sorting</u>.

New idea: break array in halves, sort each half, then merge the sorted halves together.



Total cost: $\approx n \cdot \log_2 n$

$\ll n^2$