More about recursion:

$$f_n = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ f_{n-1}+f_{n-2} & \text{else} \end{cases}$$

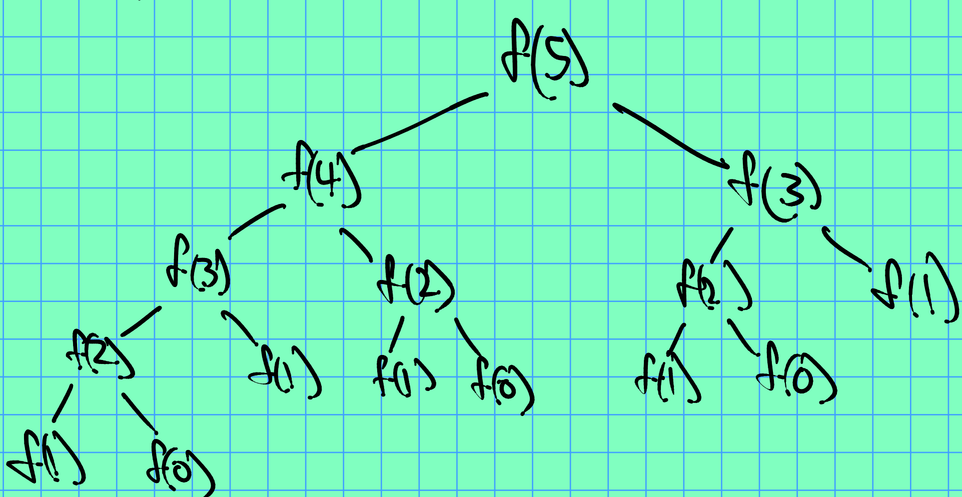
in C++:

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

Recursion Trees: graph of the function calls made during execution of a recursive function.

Each node corresponds to a function call. There is an edge $f(x) \to f(y)$ if $f(y)$ was called during execution of $f(x)$.

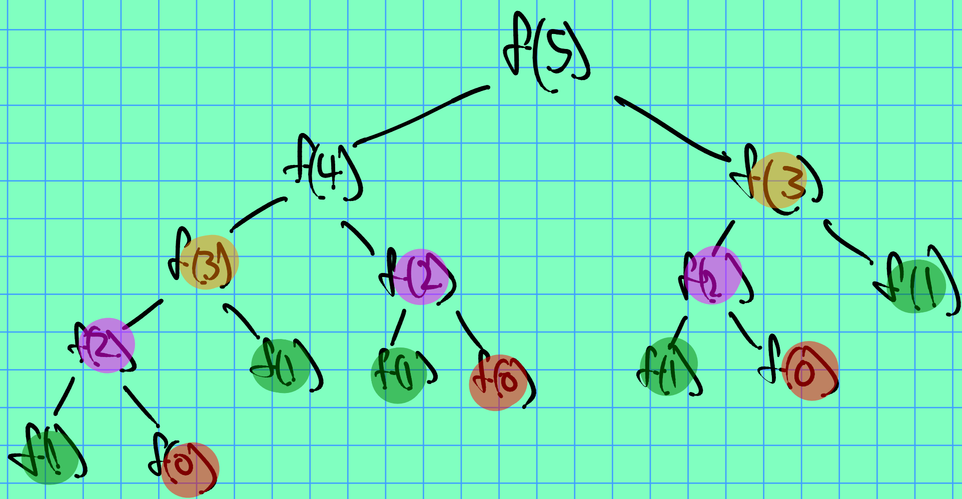
Example: draw recursion tree for $f(5)$.



As a function of n, $\approx$ how many calls does $f(n)$ require?

Answer $\approx 2^n$. x_x

Problem: lots of intermediate work is thrown away:



How to fix?, Recursive solution looked very nice, but was very slow.

Can we somehow get the best of both worlds?

New version that saves the work:

```
int f(int n, map<int, int>& M) {
    // make sure we don't already know the answer:
    if (M.find(n) != M.end())
        return M[n];
    // else compute it recursively:
    int r;
    if (n < 2) r = n;
    else r = f(n-1, M) + f(n-2, M);
    // save our work!
    M[n] = r;
    return r;
}
```

This technique is called "memoization".

```
map<int,int> fAnswers;
```

f(59, fAnswers);

More challenging problem: compute all "subsets" of a vector:

if $v = [1, 2, 3]$, output would be

$[[\,], [1], [2], [3],$
$[1,2], [1,3], [2,3]$
$[1,2,3]]$