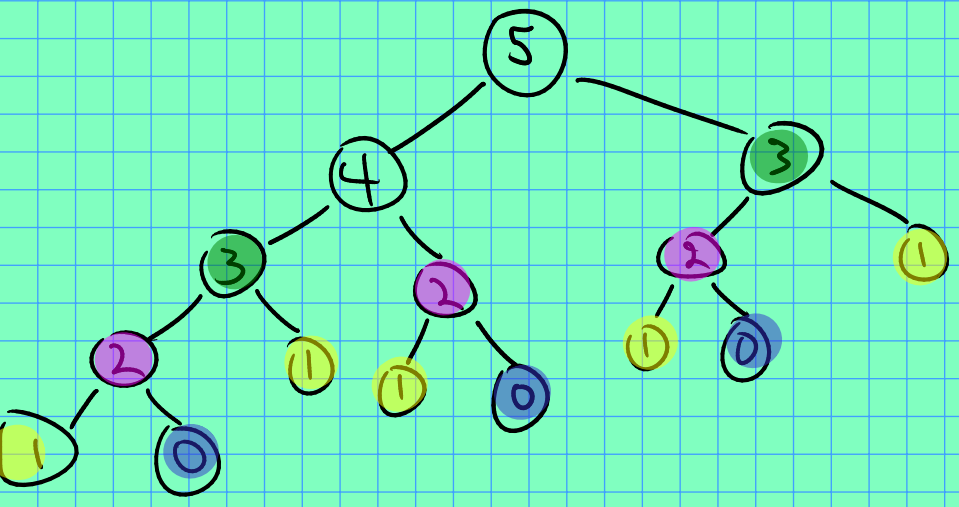
```
int fib(int n)
{
    if (n < 2) return 1;
    return fib(n-1) + fib(n-2);
}
```

Let's see what's going on by drawing the recursion tree. Say we call fib(5):



≈ How many calls on input n?

Close to 2^n. Yikes.

($\geq 2^{n-2}$?)

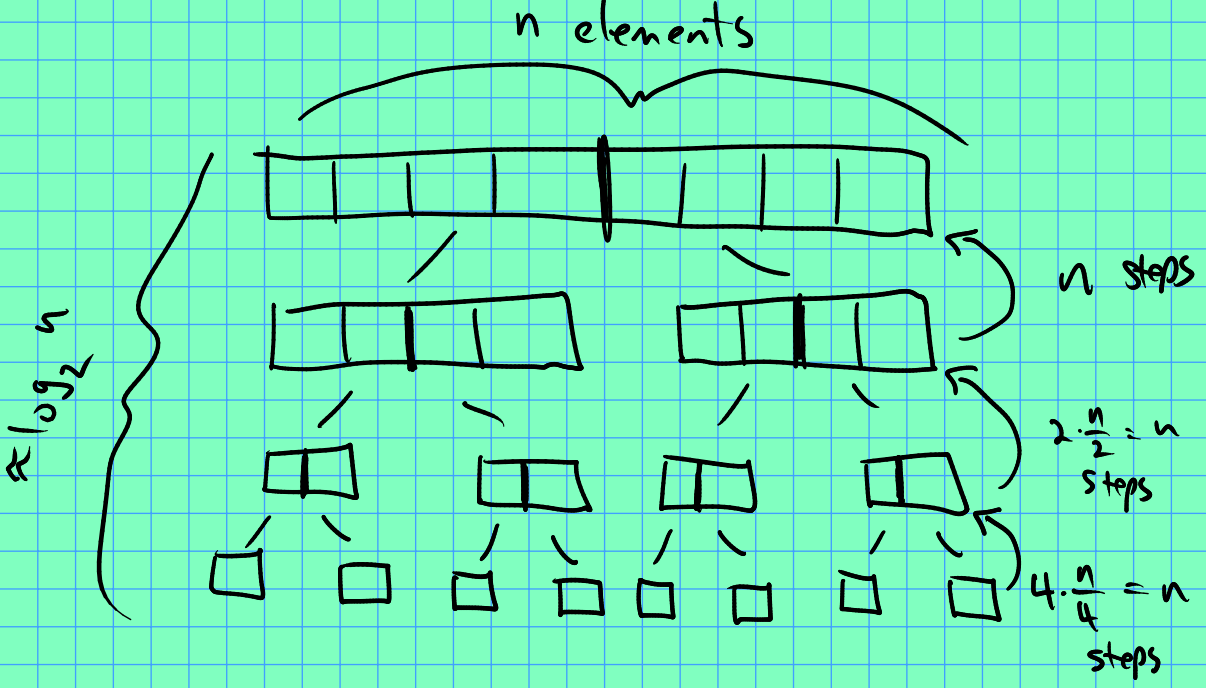
Issue: fib(n) makes many repeated calls to fib(k) for $k << n$. The function has no memory.

Possible fix: "memoization"

Just give your function memory!

```
int fibM(int n, map<int,int>& M)
{
    if(n<2) return 1;
    // check for answer to n in M
    if(M.find(n) != M.end()) // found an
        return M[n];         // entry for n
                             // in M.
    // n never before computed...
    int result = fibM(n-1,M) + fibM(n-2,M);
    M[n] = result;
    return result;
}
```

Useful example: Merge sort.



$\therefore$ total cost of sorting $\approx n \cdot \log_2 n$

Much better than selection sort!

Cost for selection sort:

$$\text{Cost} \geq n + n-1 + n-2 + \cdots + 1$$

$$= \frac{n(n+1)}{2} \geq \frac{n^2}{2} \gg n \log_2 n$$

for large n.