Q4.

```c
int fib(int n)
{
    assert(n > 0);
    if (n == 1 || n == 2) {
        return 1;
    } else {
        return fib(n - 1) + fib(n - 2);
    }
}
```

Call tree for fib(3):

fib(3)

fib(2) + fib(1) = 2

= 1   = 1

Call tree for fib(5):

fib(5)

fib(4) + fib(3) = 5

fib(3) + fib(2) = 3   fib(2) + fib(1) = 2

= 1   = 1   = 1

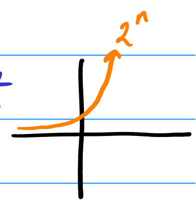
fib(2) + fib(1) = 2

= 1   = 1

Is this method feasible for, say, n = 2521?

No! We recompute fib(k) for some values of k quite a lot.

The complexity of fib(n) is actually $O(2^n)$, so slight increases in the input lead to exponential time increases



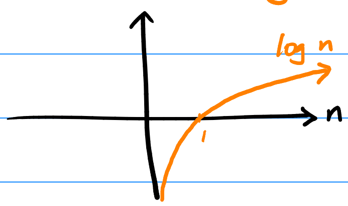
Q5.

```c
int pow(int x, unsigned int n)
{
    int res = 1;
    for (int i = 1; i <= n; i++) {
        res = res * x;
    }
    return res;
}
```

We can compute x^n in $O(n)$ time

How about in $O(\log n)$ time?

What does $O(\log n)$ time even mean?



As input grows... The amount of growth decreases.
(increasing at a decreasing rate)

Hint 1: $(x^2)^{n/2} = x^n$. (This is often called exponentiation by squaring)

Hint 2: If n is odd, $x^n = x \cdot (x^2)^{\frac{n-1}{2}}$

pow(x,n) = (you try this)