Reducing a problem X to a problem Y:

describe/design an algorithm for X that uses an algorithm for Y as a "black box" or "subroutine"

does not depend on how alg for Y works

uses only what & how fast for Y's alg

similar to proving theorems using lemmas

special reductions:
induction + recursion

Let n be a positive integer

A *divisor* d of n is an integer d s.t. n/d is an integer.

n is *prime* if it has exactly two divisors n & 1.

n is composite if more than two divisors

Thm: Every integer $n>1$ has a prime divisor.

Direct proof: Let n be an arbitrary integer greater than 1.

. . . blah blah blah . . .

Thus, n has at least one prime divisor. □

Proof by contradiction: For the sake of argument, assume there is an integer greater than 1 with no prime divisor.
Let n be an arbitrary integer greater than 1 with no prime divisor.

. . . blah blah blah . . .

But that's just silly. Our assumption must be incorrect. □

Suppose exists an integer > 1 with no prime divisor.

Let n be <u>the smallest</u> int > 1 with no prime divisor.

n divides itself, but n has no prime divisors $\Rightarrow$ n is not prime

$\exists$ a divisor d of n s.t. $1 < d < n$

n is smallest counterexample so a prime p divides d

d/p is an integer

$\Rightarrow (n/d)\cdot(d/p) = n/p$ is an integer

so p divides n $\perp$

contradiction!

Direct proof:

Let n be an arbitrary int >1.

Assume for every integer k s.t. $1<k<n$, integer k has a prime divisor.

Suppose n is prime, it is its own prime divisor.

(otherwise)

o.w. $\exists$ divisor d s.t. $1<d<n$

By assumption, d has a prime divisor p

d/p is an int $\Rightarrow (n/d)\cdot(d/p) = n/p$ is an integer

So p is a prime divisor of n

proof by induction

induction hypothesis:
assuming no strictly smaller counterexamples

base cases: argue directly without using hypothesis
(n is prime)

inductive cases: the other cases

Theorem: $P(n)$ for every positive integer n.

Proof by induction: Let n be an arbitrary positive integer.

Assume that $P(k)$ is true for every positive integer $k < n$.

There are several cases to consider:

- Suppose n is *... blah blah blah ...*
  Then $P(n)$ is true.
- Suppose n is *... blah blah blah ...*
  The inductive hypothesis implies that *... blah blah blah ...*
  Thus, $P(n)$ is true.

In each case, we conclude that $P(n)$ is true. □

recipe: 1) write down template

2) think <u>big</u>

<u>trust</u> claim is true for $k<n$

3) look for holes (base cases)

4) rewrite everything

DO NOT:

1) Do not make the hypothesis for $k = n-1$ only.

2) Do not assume for n & prove for $n+1$.

don't build up, instead reach down

Recursion: Given a problem instance.

1) try to reduce it to one or more simpler instances of the same problem (smaller problem size)

2) if you can't reduce, solve instance diretly (base cases)

Trust the Recursion Fairy can solve the simpler instances.

peasant multiplication:

$$x \cdot y = \begin{cases} 0 & \text{if } x = 0 \\ \lfloor x/2 \rfloor \cdot (y+y) & \text{if } x \text{ is even} \\ \lfloor x/2 \rfloor \cdot (y+y) + y & \text{if } x \text{ is odd} \end{cases}$$

just need to know addition & halving

```
PEASANTMULTIPLY(x, y):
    if x = 0
        return 0
    else
        x' ← ⌊x/2⌋
        y' ← y + y
        prod ← PEASANTMULTIPLY(x', y')    ⟨⟨Recurse!⟩⟩
        if x is odd
            prod ← prod + y
        return prod
```

Correctness of recursive algorithms follows from induction!

Given x & y.

If $x=0$, $x \cdot y = 0$ ✓

o.w. $x' = \lfloor x/2 \rfloor < x$.

Assume $PM(k, y)$ is correct for $0 \leq k < x$.

So $PM(x', y')$, it is correct.

Rest of alg follows formula.