In CS, we're concerned w/ solving problems w/ a computer.

A problem for a computer must be defined precisely + unambiguously by its input and its desired output.

ex Sort an array.
input: array + way to compare elements
output: sorted array

compute the factorial of a pos. int
input: $n \in \mathbb{Z}^+$
output: $n!$

Note that we need the tools of discrete math to define these inputs + outputs precisely!

A solution is some method of taking in an arbitrary input and computing an output w/ desired properties defined by the problem.

Typically this method is an algorithm, a sequence of steps you can perform to get from input to output.

In practice, we write an algorithm as a mix of precise + unambiguous notation and some words to give intuition. We call this mix pseudocode.

ex fact(n):
    if $n=1$ then
        return 1
    else
        → return $n \cdot$ fact$(n-1)$

pseudo-code for a recursive factorial algorithm

For any algorithm, you should ask yourself:

1. Does the alg work? Does it give the correct output for every valid input?

Proving ↗ a focus of later classes...
But for some algs, we can do this.

ex The recursive factorial alg is correct, i.e., computes $n!$ $\forall n \geq 1$.

Pf For pos. int. n, let $P(n)$ denote the property that fact$(n) = n!$ We prove by mathematical induction on n.

base case $(n=1)$ fact(1) returns 1.

inductive case: we WTS $\forall n \geq 2 : P(n-1) \Rightarrow P(n)$.

Assume $P(n-1)$. That is, fact$(n-1)$ returns $(n-1)!$ WTS fact(n) returns $n!$

fact(n) = n fact$(n-1)$	by def. of fact alg.
$= n(n-1)!$	by IH
$= n!$	def. of !

∎

2. Does the algorithm work efficiently?

ex for array sorting problems

```
sort (A) :
    let S = the set of all orderings of A's elts
    for x in S :
        if x is sorted then
            return x
```

Is this algorithm efficient? No — if elts of A are distinct, $|S| = \text{length}(A)!$

We focus on runtime.

<u>How do we measure runtime?</u>

idea #1: implement alg, run it, time it...
- depends on software, hardware, OS
- implementation takes time, error prone
- what input do we run it on?

idea #2: 1 find a function that expresses the alg's <u>runtime</u> as a function of input size

→ # of primitive ops: arithmetic ops, logical ops, variable retrieval + assignment

2 Use big O to represent the function, so we get a big-picture idea + can compare to other algs

examples

for $i=1$ to $i=n$ do
  for $j=1$ to $j=n$ do
    $\underbrace{sum}\ \underbrace{=}\ \underbrace{sum}\ \underbrace{+}\ \underbrace{i \cdot j}$] 4? 6? doesn't matter $O(1)$] $\sum_{j=1}^{n} 1 = n$] $\sum_{i=1}^{n} n = n^2$

① figure out how many primitive ops

② express in big O : $f(n) = n^2 = O(n^2)$.

ex for $r=1$ to $r=n$ do
  for $c=1$ to $c=m$ do
    $p[r][c] = r+c$] $O(1)$] $\sum_{c=1}^{m} 1 = m$] $\sum_{r=1}^{n} m = nm$

① $f(n,m) = nm$

② $nm = O(nm)$

ex for $x=1$ to $x=n$ do
  for $y=1$ to $y=n$ do
    foobar()] $O($runtime of foobar$)$

② $O(n^2 \cdot$ runtime of foobar$)$

ex for $i=1$ to $i=n$ do
  for $j=i$ to $j=n$ do
    $sum = sum + i \cdot j$] $O(1)$] $\sum_{j=i}^{n} 1 = n-i+1$]

$\sum_{i=1}^{n} (n-i+1)$

$= n^2 - \frac{n(n+1)}{2} + n$

① $f(n) = n^2 - \frac{n(n+1)}{2} + n$

② $f(n) = O(n^2)$

$$\sum_{i=1}^{n} (n-i+1) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1$$

$$= n^2 - \frac{n(n+1)}{2} + n$$

hard one...