Bio. algs

Intro:

Today

- Syllabus & overview
- A little about me
- Some questions for you all...

What *is* an algorithm?

First, a question:

What CS background do you all have?

Languages:

Classes:

Intro:

data structures:

discrete math:

Algorithm: (Ch 1 of text)

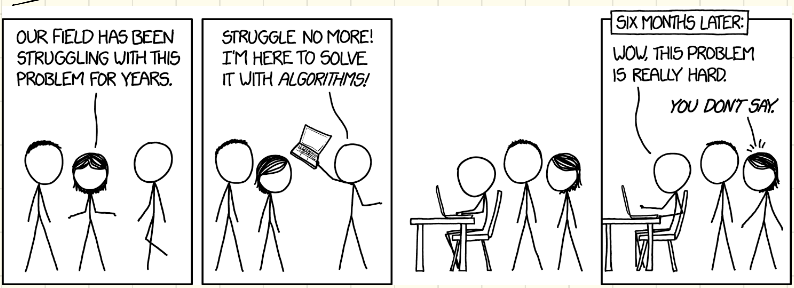
Something in between an English prose description and an implementation.

Why?

Goal:

Why (for biologists)?

Caveat:



Algorithm (defn):

A sequence of instructions to perform in order to solve a well-specified problem.

Well-specified problem (dfn):

Given specified inputs & outputs, an unambiguous mapping from inputs to acceptable outputs.

All a bit hazy, so let's consider some examples...

Pseudo code:

Formally, CS people write pseudo code to specify algorithms.

Variable assignment: $a \leftarrow 12$

Arithmetic: +, −, *, /

Conditional:
```
if A
    B
else
    C
```

also: procedures

Ex:
```
MAX(a,b)
  if a<b
      return a
  else
      return b
```

Array access: a_i (or $A[i]$)

Loops: ① for i ← a to b
{CODE Here}

ex: SUMINTEGERS(n)

```
sum ← 0
for i ← 1 to n
    sum ← sum + i
return sum
```

② while (A)
{CODE here}

ex: ADDUNTIL(b)

```
i ← 1
total ← i
while (total ≤ b)
    i ← i + 1
    total ← total + i
return i
```

Correctness + runtime

Our 2 core concerns in this class:

1. Correctness

   How do you know the algorithm works on <u>all</u> inputs?

2. Efficiency

   When is one algorithm better than another?

An example (Sec. 2.2 + 2.3 in book)

How do you make change?
(using as few coins as possible)

Algorithm:

Is this correct?

Consider the US system ~150 years ago:

Coins: 25¢
20¢
10¢
5¢
1¢

Will our algorithm work?

A more interesting example:
the stable marriage problem.

Q: How do doctors get matched to internships?

Dfn: We say a doctor-hospital matching is <u>unstable</u> if:

- doctor a is assigned to hospital A
- doctor b is assigned to hospital B
- and a prefers B & B prefers a

In other words, doctor a & hospital B would <u>both</u> be happier with each other than with current assignments.

Resident matching problem:

Given residents (w/ preferences) & hospitals (w/ preferences), compute a matching that is <u>stable</u>.

Solution: "Boston pool" algorithm
or Gale-Shapley algorithm

Repeat in rounds:

1. An arbitrary unassigned hospital H offers a position to the best doctor d (who has not yet said no)

2. Each doctor d ultimately accepts the best offer they get.

   So: if she doesn't have a better offer in hand, d tentatively accepts H.

   If d has an offer but prefers H, rejects prior offer.

   Otherwise, d rejects H.

Example: Dr. Quincy
Dr. Rotwang
Dr. Shepherd
Dr. Tam

Hospitals: Arkham Asylum
Bethlem Royal Hospital
County General Hospital
Dharma Initiative

q	r	s	t
A	A	B	D
B	D	A	B
C	C	C	C
D	B	D	A

A	B	C	D
t	r	t	s
s	t	r	r
r	q	s	q
q	s	q	t

Given these preferences as input, the Boston Pool algorithm might proceed as follows:

1. Arkham makes an offer to Dr. Tam.
2. Bedlam makes an offer to Dr. Rotwang.
3. County makes an offer to Dr. Tam, who rejects her earlier offer from Arkham.
4. Dharma makes an offer to Dr. Shephard. (From this point on, because there is only one unmatched hospital, the algorithm has no more choices.)
5. Arkham makes an offer to Dr. Shephard, who rejects her earlier offer from Dharma.
6. Dharma makes an offer to Dr. Rotwang, who rejects her earlier offer from Bedlam.
7. Bedlam makes an offer to Dr. Tam, who rejects her earlier offer from County.
8. County makes an offer to Dr. Rotwang, who rejects it.
9. County makes an offer to Dr. Shephard, who rejects it.
10. County makes an offer to Dr. Quincy.

Result: $(A,s), (B,t), (C,q), (D,r)$
Stable?

Correctness?

Well: the algorithm continues as long as any hospital is empty.
So when it terminates, every position is filled.

Suppose a is assigned to A, but prefers B.

- we know a got no better offer

so:

Efficiency (2.7 & 2.8 in book)

- Exact speed can depend on many variables besides the algorithm.

Issues at play:

Alternative approach:

Count **primitive operations**, which are smallest operations.

In addition: generally only examine **worst case** running time.

Why?

<u>Now</u>: How to actually compare?

- Remember small difference may be due to processor, language, or any number of things that aren't dependent on the algorithm.

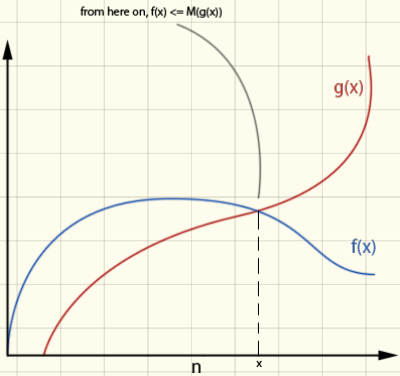
- Also: need a way to account for inputs changing

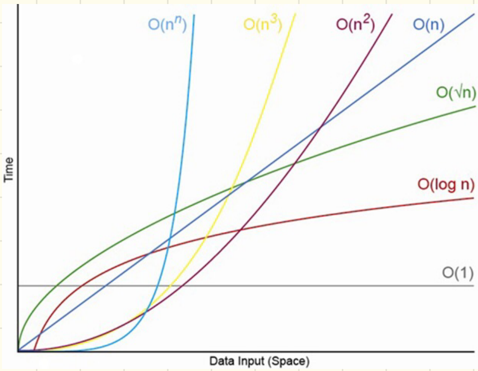
  eg searching a list

Big-O notation

We say $f(n)$ is $O(g(n))$ if $\forall n > n_0$, $\exists\, c > 0$ such that

$$f(n) \leq c \cdot g(n)$$





Common runtimes

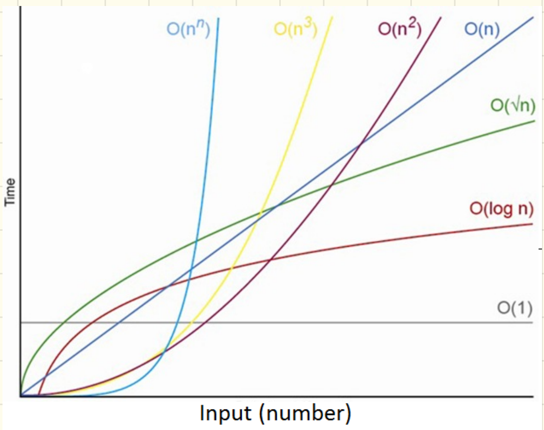
1. $O(1)$
2. $O(\log n)$
3. $O(n)$
4. $O(n \log n)$
5. $O(n^2)$

   ⋮

   (polynomial)

And: $O(2^n)$

$O(n!)$



Next time:

- Connecting run times & algorithms

Also: Stay tuned for HW 1!