CS 331, Fall 2024
Lecture 1 (8/26)

Today:
- Logistics
- Background
  - Induction
  - Asymptotics
  - Data Structures
- Orders of magnitude
- Recursion
- Multiplication

Logistics

We'll use the following websites.

1) Ed
- Ask questions about lecture, HW.
- Announcements about schedule changes.

2) Canvas
- Turn in HW, grades released.

3) Class website
- kjstian.github.io/CS331.html
- Lecture notes, HW posted here.
- Syllabus, course information.
- Feedback forms!

Grade breakdown

40% HW (6, lowest dropped)

30% Midterms (2)

30% Final exam

Lecture feedback

Let us know what made the least sense. We'll go over in discussion section.

Notes feedback

Please report any typos, confusing examples.

Things to note:

- No AI tools on HW!

  We want you to have good practice opportunities for tests, and also to learn the material better.

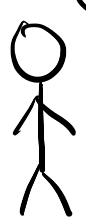
- Coding component.

  HW will have ≤ 1 coding question in Python.

  Hands on experience using algorithms! Let me know <u>early</u> if concerns about coding background.

Syllabus: Algorithms "starter pack"

You are here



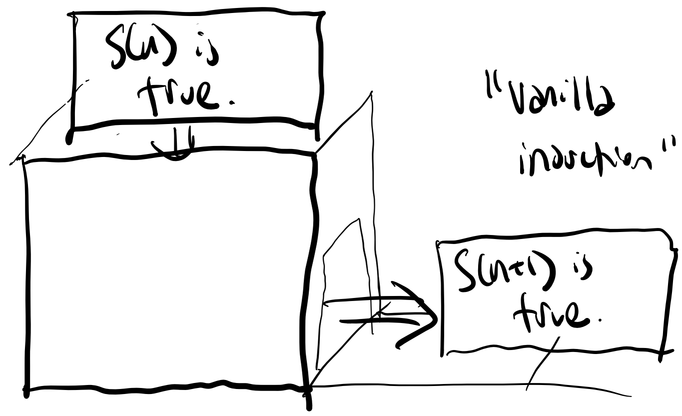
Part I: Paradigms	Part II: Toolkits	Part III: Hardness	The wilderness...
- Recursion - Dynamic Programming - Greedy algorithms	- Graph algorithms - Continuous algorithms - Randomized algorithms	- Complexity theory	- Interview - Job - Research - ...
Common strategies for designing + analyzing algorithms.	Powerful techniques for specialized problems.	When is it not possible to design an algorithm?	

Background: Induction (Part I, Section 2)

Suppose you want to prove some statement $\boxed{S(n)}$ for all $n \in \mathbb{N}$. Induction builds "machines" to generate new true statements out of old ones. E.g.

$\boxed{S(1) \text{ is true.}}$

base case



"Vanilla induction"

induction machine

This is enough to prove all $S(n)$! We can feed $S(1)$ into the machine to get $S(2)$, feed it in again to get $S(3)$, etc.

Example

Claim: For all $x \in \mathbb{R}$, $n \in \mathbb{N}$,

$$x^n - 1 = (x-1)\left(\sum_{i=0}^{n-1} x^i\right).$$

Proof: Base case $n=1$: $x - 1 = (x-1)\left(\sum_{i=0}^{0} x^i\right) = x-1$ 　$S(1)$ is true.

Induction machine: Suppose $x^n - 1 = (x-1)\left(\sum_{i=0}^{n-1} x^i\right)$

Then, $x^{n+1} - 1 = (x^{n+1} - x^n) + (x^n - 1)$

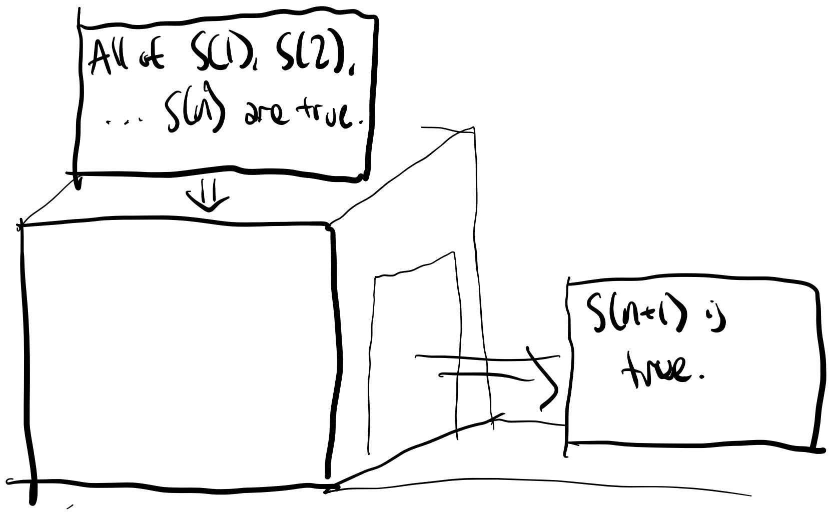
$$= (x-1)x^n + (x-1)\left(\sum_{i=0}^{n-1} x^i\right)$$

$$= (x-1)\left(\sum_{i=0}^{n} x^i\right). \blacksquare$$

$S(n)$ implies $S(n+1)$.

There are other "induction machines" that allow proving all S(n).

Strong induction machine



In math, as long as you can prove all S(n), you are happy. You can use as many machines as you want.

In algorithms, we care about efficiency along with correctness. It matters how many times you use the machine.

Background: Asymptotics (Part I, Section 3)

In this class, "n" means the size of the input.

How to compare Algo 1, solves problem in $f(n)$ time
Algo 2, solves problem in $g(n)$ time?

- When n is small, Algo 1 & 2 both fast.
- We care about performance on big inputs.
- Asymptotics: how do f, g compare as $n \to \infty$?

Assume: for a constant n_0, $f(n), g(n)$ are positive for $n \geq n_0$.

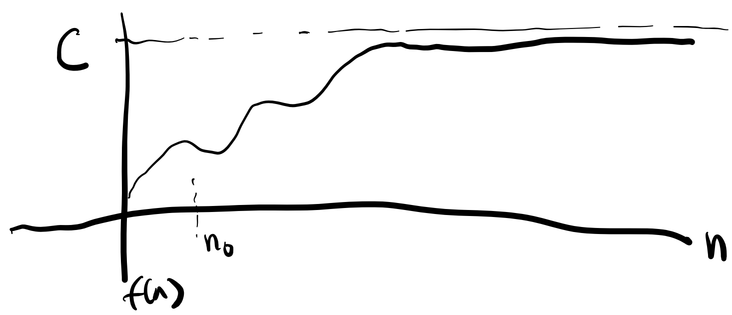
The asymptotics zoo:

$$f(n) = \begin{matrix} O(g(n)) \\ \Omega(g(n)) \\ \Theta(g(n)) \\ o(g(n)) \\ \omega(g(n)) \end{matrix} \iff \frac{f(n)}{g(n)} = \begin{matrix} O(1) \\ \Omega(1) \\ \Theta(1) \\ o(1) \\ \omega(1) \end{matrix}$$

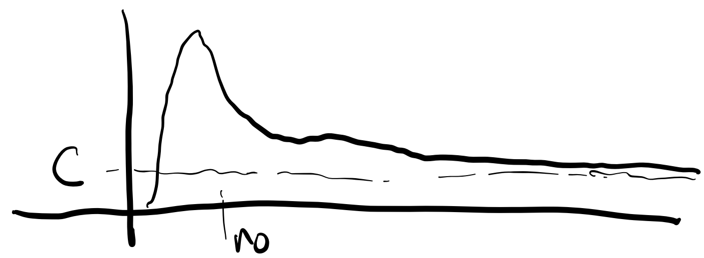
Suffices to understand.

When is a function $f(n)$...

... $O(1)$? There's some constant C s.t. $f(n) \le C \cdot 1 = C$, $\forall\ n \ge n_0$.



... $\Omega(1)$? There's some constant C s.t. $f(n) \ge C \cdot 1 = C$, $\forall\ n \ge n_0$.



... $\Theta(1)$? It just means $O(1)$ and $\Omega(1)$.

... $o(1)$? It just means not $\Omega(1)$.
AKA there's no valid lower bound $\forall n \ge n_0$.
AKA $\lim_{n \to \infty} f(n) = 0$.

... $\omega(1)$? It just means not $O(1)$.
AKA no valid upper bound.
AKA $\lim_{n \to \infty} f(n) = \infty$.

Background: Data Structures (Part I, Section 7)

This is assumed knowledge from prereqs.
However, we give full descriptions for review in the notes.

Please make sure you are familiar! Very useful in algorithms.

General rule: You can cite anything from the notes as true, without proof. Feel free to use data structure APIs "for free".

Array
- Init: $O(1)$
- Insert: $O(1)$
- Delete: $O(1)$
- Query: $O(1)$

Heap
- Init: $O(n)$
- Insert: $O(\log(n))$
- ExtractMin: $O(\log(n))$
- Delete: $O(\log(n))$

LinkedList
- Init: $O(1)$
- Insert After: $O(1)$ (with address)
- Delete: $O(1)$ (with address)
- Query: $O(1)$ (with address)

BST
- Init: $O(1)$
- Insert: $O(\log(n))$
- Delete: $O(\log(n))$
- Query: $O(\log(n))$
- Search: $O(\log(n))$
- Index: $O(\log(n))$

HashTable
- Init: $O(1)$
- Insert: $O(1)$
- Search: $O(1)$
- Delete: $O(1)$

only in expectation.

Stack/Queue
- Based on LinkedList

Orders of magnitude (Part II, Section 4)

From lemma 11, part I: $\log^a(n) = o(n^b)$ $\forall a, b > 0$.

Example: $\log^{100}(n) = o(n^{0.1})$.

Lemma 12, Part I: $n^a = o(b^n)$ $\forall a > 0, b > 1$.

Example: $n^{100} = o(2^n)$.

Punchline: asymptotically,

polylog $\ll$ polynomial $\ll$ exponentials.

Taxonomy of runtimes

fast $\longleftrightarrow$ slow

Sublinear algos

$f(n) = o(n)$

only in Part VII

polynomial-time algos

$n < n\log(n) < n^2 < n^3 < \ldots$

Focus for most of class.

exponential-time algos

$f(n) = c^n$, $c > 1$

only in Part VIII

e.g. $n = 4 \cdot 10^9$ Google searches every day.

- We care about log factors. $\log^3(n) > \sqrt{n}$ if $n \le 10^7$.
- We'll be up front when constant factors are large.

e.g. if $c = 2$, $n = 250$, $c^n >$ # particles in the universe.

Recursive algorithms (Part II, Section 1)

To analyze an algorithm:
1) Proof of correctness
2) Analyze runtime (next lecture)

How to analyze correctness?

```
ComplexAlgo(inputs):
    SimpleAlgo1(inputs')
    SimpleAlgo2(inputs'')
    ...
```

Step 1: analyze each subroutine. are they all correct?

Step 2: analyze "stitching together" of subroutines. does it solve overall problem, assuming correct subroutines?

Key idea of recursion: Use algo itself as subroutine.

```
RecursiveAlgo(size n input):
    RecursiveAlgo(size n' < n input)
    RecursiveAlgo(size n'' < n input)
    ...
```

~~Step 1:~~ Subroutines are correct for free! Thanks to the "recursion fairy" ☺

Step 2: Still need to do this.

Why can we believe in the recursion fairy?

Strong induction.

Multiplication (Part II, Section 2)

Basic model: add, subtract, multiply 1-digit #s in $O(1)$ time.

__Warmup__: How expensive is adding n-digit #s?

$$\boxed{\text{Runtime: } O(n)}$$

$$\begin{array}{rr} & \overset{1\,1\,1\,1\,1}{a = 123456} \\ + & b = 987654 \\ \hline & 1111110 \end{array}$$

"grade-school addition"

each of n outputs needs $O(1)$ time.

$$\begin{array}{rr} & a = 123456 \\ \times & b = 987654 \\ \hline & 493824 \\ & 617280 \\ & 740736 \\ & 864192 \\ & 987648 \\ + & 111104 \\ \hline & 121931812224 \end{array}$$

"grade-school multiplication"

$$\boxed{\text{Runtime: } O(n^2)}$$

n numbers to add
$O(n)$-digits long each.

Can recursion help? (Does solving smaller problems make progress towards overall goal?)

Idea 1: divide-and-conquer

$$a = a_1 \cdot 10^{n/2} + a_0$$
$$b = b_1 \cdot 10^{n/2} + b_0$$

e.g.

$\underbrace{123}_{a_1}\ \underbrace{456}_{a_0}$

$\underbrace{987}_{b_1}\ \underbrace{654}_{b_0}$

How to use smaller subproblems recursively?

$$ab = a_1 b_1 \cdot 10^n + \underbrace{(a_1 b_0 + a_0 b_1)}_{\text{"middle coefficient"}} \cdot 10^{n/2} + a_0 b_0$$

to compute... need 4 $\frac{n}{2}$-digit multiplies + $O(n)$ extra work.

Let $T(n)$ = cost of multiplying n-digit #s. We showed:

$$T(n) = 4T\left(\tfrac{n}{2}\right) + O(n).$$

(next lecture, we'll show this is $T(n) = O(n^2)$. no savings!)

Idea 2: Karatsuba recursion

- Still compute $a_1 b_1$ and $a_0 b_0$ (2 subproblems).
- Observe that "middle coefficient" can reuse these!

$$a_1 b_0 + a_0 b_1 = \underbrace{(a_1 + a_0)(b_1 + b_0)}_{\text{(1 subproblem)}} - a_1 b_1 - a_0 b_0$$

$$T(n) = 3T\left(\tfrac{n}{2}\right) + O(n).$$

(next lecture, we'll show this is $T(n) = O(n^{1.58\ldots})$)