CS 331, Fall 2026
Lecture 1 (8/24)

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

Logistics

We use the following websites.

1) Course website (kjtian.github.io/cs331_fa26.html)
   - Lecture notes, HW posted
   - Syllabus, course info
   - Feedback forms

Lecture feedback
What made the least sense?
TAs will go over in sections.

Notes feedback
Typoes, confusing examples

2) Ed
   - Announcements, practice tests
   - Ask questions about lecture, HW

3) Canvas

- HW turn-in, grades released
- Lecture videos (mini-lectures on website 3:45-4:30)

GDC 4.516

$\boxed{\text{Philosophy}}$ Confusion is good!

Getting unconfused ⇒ Learned something.

How can you tell? Explain / discussion!

Many opportunities

- Class
- Discussion section
- Office hours

} incentives!

$\boxed{\text{AI Policy}}$ My job:

- help you learn
- make sure you learn

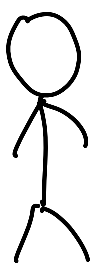
Not OK: Anything for a grade. HW, quiz, exam

OK: Anything else. Explain lecture notes

Generate practice questions

Grading

30%	HW	6x, 5 problems each
		$\approx$ 1 coding question, NO AI
Bonus:		Lowest dropped
20%	quizzes	10x, 2 problems each (pick 1)
		first 10 min of lecture
Bonus:	up to 5!	Lowest 2 dropped
		1/3 discussions attended = +1, 2/3 = +2
		Open OH 2–5, 9/4 or 9/11 = +1
30%	midterms	5 problems each (pick 3)
20%	final exam	9 problems (test bank, pick 6)



You are here

Paradigms	Toolkits	Hardness	Wilderness
– Recursion	– Graphs	– Complexity	– Interview
– Dynamic programming	– Continuous		– Job
– Greedy	– Randomized		– Research
			– ...

Background: Induction (Part I, Section 2)

Mini-lecture today!

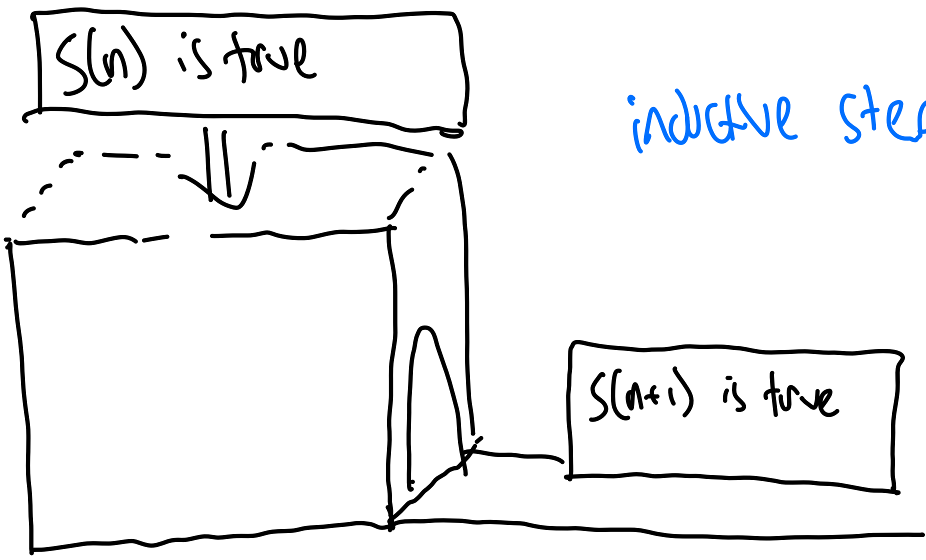
You want to prove a statement $S(n)$ for all $n \in \mathbb{N}$.

e.g. $S(n)$: There are 2^n subsets of $[n] = \{1, 2, \ldots, n\}$

<u>Induction</u> builds "machines" to generate new statements out of old ones. Example: "basic induction"

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

②



inductive step

This can prove any $S(n)$: $S(1) \Rightarrow S(2) \Rightarrow S(3) \ldots \Rightarrow S(100) \ldots$

Ex. $S(1): |\{\emptyset, \{1\}\}| = 2 = 2^1$ ✓

$S(n+1)$: Subsets of $[n+1]$ containing $n+1$ 2^n

not containing $n+1$ 2^n ✓

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

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

Proof: Base case $S(1)$: $x - 1 = (x-1)(1)$ ✓

Induction Assume $S(n)$ is true, so we know

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

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

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

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

Consequence: $a_0 + a_0 r + a_0 r^2 + \dots + a_0 r^k$

$$= a_0\left(1 + r + r^2 + \dots + r^k\right) = a_0 \cdot \frac{r^{k+1} - 1}{r - 1}$$

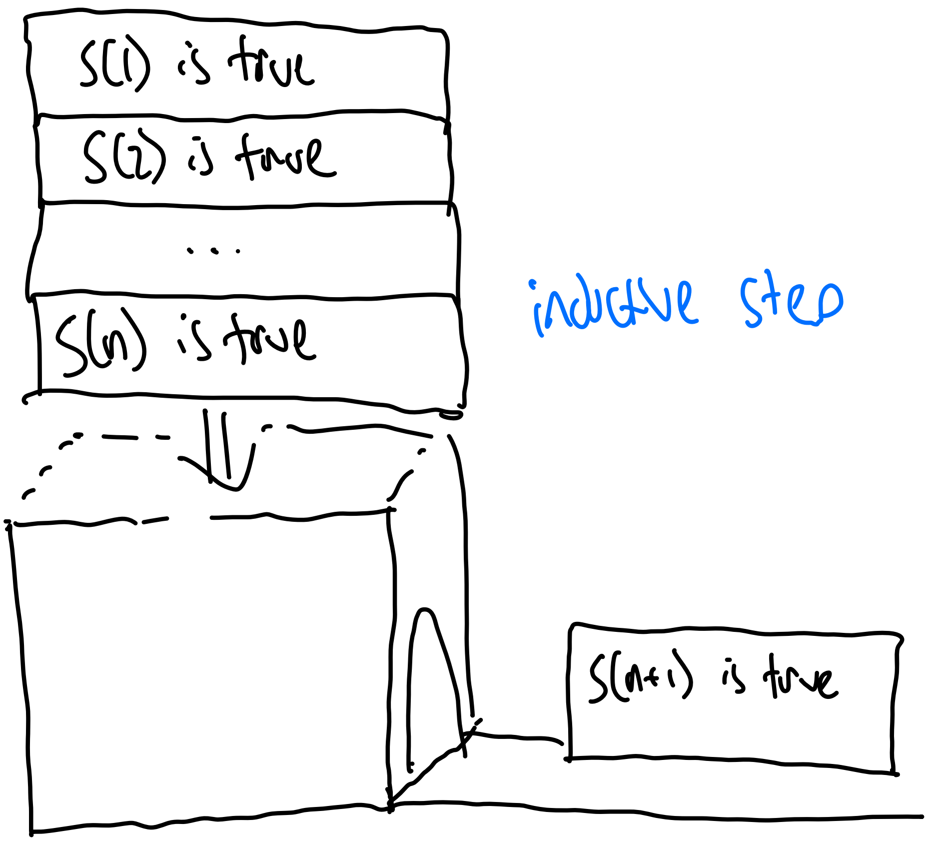
Sum of geometric series. $(r \neq 1)$

Another version: Strong induction.

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

We get this stronger assumption for free during normal induction!

2



inductive step

Still proves all $S(n)$.

$\boxed{\text{Example}}$ Claim: All $n \geq 2$, $n \in \mathbb{N}$ can be written as a product of primes.

Proof: Base case $S(2)$: $2 = 2$ ✓

Induction If $n \geq 2$, either...

n is prime $n = n$ ✓

n is composite $n = \underbrace{a}_{p_1 p_2 \cdots p_i} \cdot \underbrace{b}_{q_1 q_2 \cdots q_j}$ ✓

strong induct:

Background: Asymptotics (Part I, Section 3)

In <u>math</u> as long as you can prove all $S(n)$, happy.

You can use as many steps as you want.

In <u>algos</u> we care about <u>efficiency</u> along with <u>correctness</u>

In this class, n = size of input to problem

How to compare Algo 1: solves in $f(n)$ time

Algo 2: solves in $g(n)$ time

$\boxed{\text{Philosophy}}$ If n is small, Algo 1, 2 both fast

We care about performance as $n \to \infty$

"asymptotics"

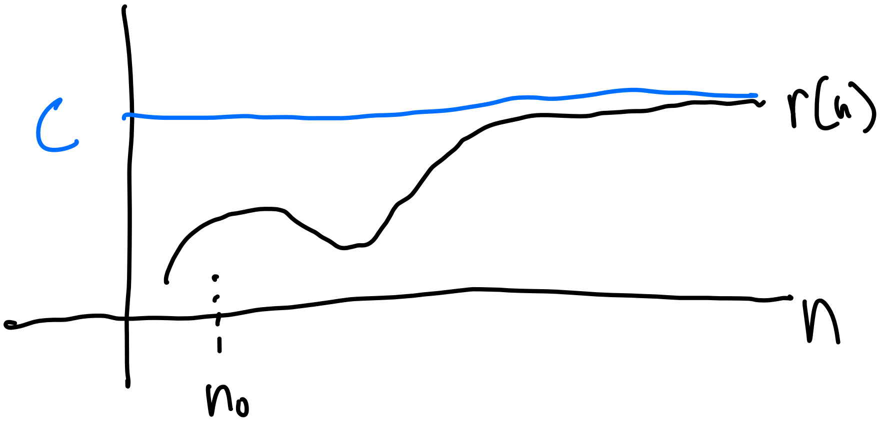
The zoo:

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

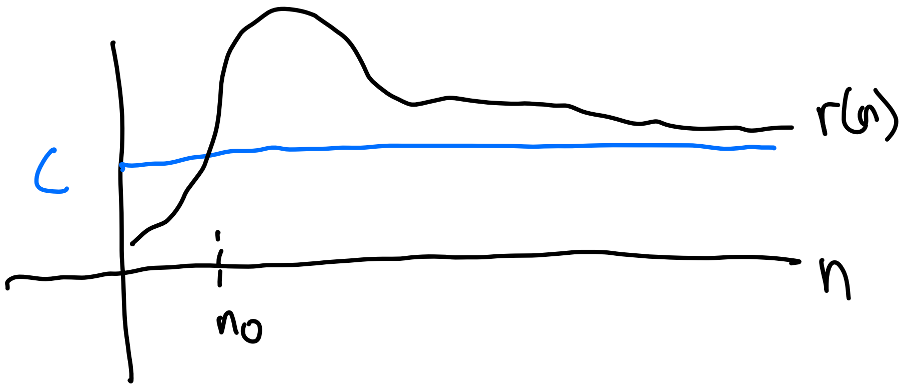
When is a function $r(n)$ ...

... $O(1)$? $\forall n \geq n_0, \; r(n) \leq C$

constant independent of n, like 100



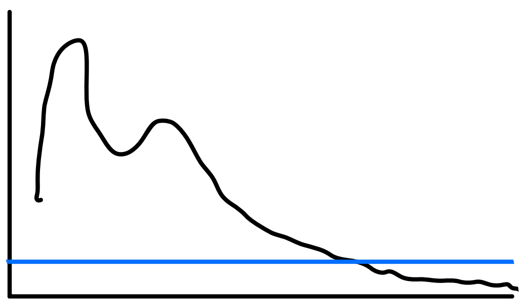
... $\Omega(1)$? $\forall n \geq n_0, \; r(n) \geq C$



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

... $o(1)$? It just means not $\Omega(1)$.

AKA $\lim_{n \to \infty} r(n) = 0$



... $\omega(1)$? It just means not $O(1)$

AKA $\lim_{n \to \infty} r(n) = \infty$

Three main function types in this class:

Exponential $\left(2^{100n},\ e^n,\ \ldots\right)$

Polynomial $\left(n^6,\ 12n^3,\ n^2+3,\ \ldots\right)$

Polylogarithmic $\left(\log^3 n,\ 3\log^4 n,\ \log n+2,\ \ldots\right)$

Helpful rule

Exponential >> Polynomial $\quad c^n = \omega(n^b) \quad \forall\, c>1,\ b>0$

Polynomial >> Polylogarithmic $\quad n^b = \omega\left(\log^a(n)\right) \quad \forall\, b>0,\ a>0$

$\boxed{\text{Exercise}}$ Rank the following asymptotically

$n^{100}e^n,\ 100000,\ \log^{1000}(n),\ n^{.001},\ n^2 3^n$

Background: Data Structures (Part I, Section 7)

Assumed background, notes provide proofs.

Make sure you're comfortable: <u>Very useful</u> in algos.

General rule: In this class, you can use anything in notes as true without proof. Use following APIs "for free"

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

Heap
- Init $O(n)$
- Insert $O(\log(n))$
- Extract Min $O(\log(n))$
- Delete $O(\log(n))$

LinkedList
- Init $O(1)$
- Insert $O(1)$ (changes index)
- Delete $O(1)$ (changes index)
- Query $O(1)$ (needs address)

BST
- Insert $O(\log(n))$
- Delete $O(\log(n))$
- Query $O(\log(n))$
- Search $O(\log(n))$

Hash Table

not used until Part VII

- Init $O(1)$
- Insert $O(1)$ (only in expectation ...)
- Search $O(1)$
- Delete $O(1)$

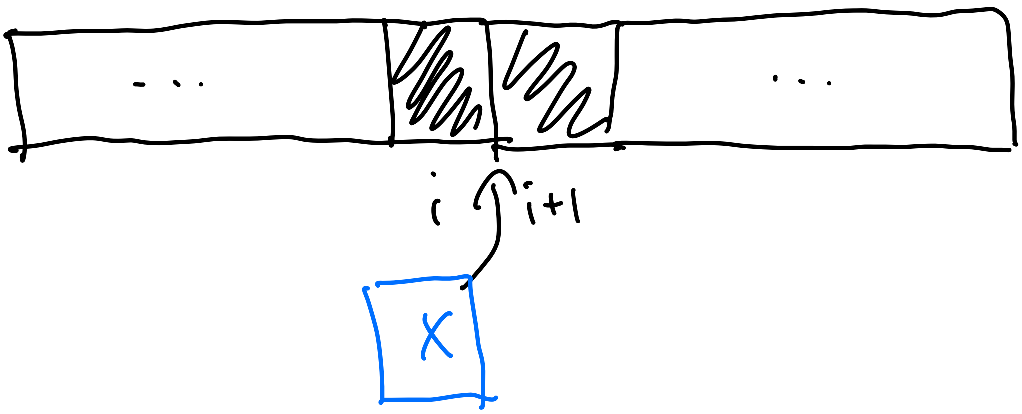
Stack/Queue

Based on LinkedList

Exercise Which is preferable?

Array vs. Linked List

- You want to insert in between adjacent elements



- You want to query the i^{th} slot in the list

Heap vs. BST

- You want to find the i^{th} largest element of unsorted list

14	-3	200	0.5	0.1

$i = 3$ Ans: 0.5

- You want to perform a task for many ($\gg n$) values of i, possibly different

Orders of magnitude (Part II, Section 4)

Taxonomy of runtimes:

polylog $\ll$ (nearly-) linear $\ll$ polynomial $\ll$ exponential

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

Sublinear algos

$f(n) = o(n)$

Part VII only

e.g. $n = 4 \cdot 10^9$

Google searches every day

We will care about log factors in this class.

Caveat: what constants?

$\log^3(n) > \sqrt{n}$ if $n \lesssim 10^7$

We will be upfront about when constants are large, usually no problem.

exponential-time algos

Characteristic of "brute-force" solution

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

Part VIII: when is there nothing better to do?

e.g. $c = 2$

$n = 250$

$c^n >$ # particles in universe

Recursive algorithms (Part II, Section 1)

To analyze an algo: 1) Proof of correctness

2) Analyze runtime (next lecture)

How to analyze correctness?

```
ComplexAlgo(inputs)
    Step 1
    Step 2
    Subroutine_1(inputs')
    Subroutine_2(inputs'')
    Step 3
    ...
```

1) Assume all subroutines are correct. Are they "stitched together" correctly?

2) Are all the subroutines correct? Prove it.

e.g. to get ready for class,

put on clothes

pack bag

start coffee machine

...

This unit's theme: particular type of algo

$\boxed{\text{Recursion}}$ key idea: only use algo itself as subroutine

RecursiveAlgo (size n input):

Step 1

Step 2

RecursiveAlgo (size $n' < n$ input)

Step 3

RecursiveAlgo (size $n'' < n$ input)

...

~~2) Are all the subroutines correct? Prove it.~~

no need: strong induction!

AKA "recursion fairy"

e.g. to paint a fence, paint left half, paint right half

Recursive algorithm is like an efficient proof by strong induction.

Multiplication (Part II, Section 2)

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

Warmup How expensive is adding n-digit #s?

Runtime: $O(n)$

$$\begin{array}{rcccccccc} & a = & 1 & 2 & 3 & 4 & 5 & 6 \\ + & b = & 9 & 8 & 7 & 6 & 5 & 4 \\ \hline & = & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{array}$$

grade-school

also: each output takes $O(1)$

How about multiplication?

$$\begin{array}{rccccccccccccc} & & & & & & & a = & 1 & 2 & 3 & 4 & 5 & 6 \\ & & & & & & \times & b = & 9 & 8 & 7 & 6 & 5 & 4 \\ \hline & & & & & & & 4 & 9 & 3 & 8 & 2 & 4 \\ & & & & & & 6 & 1 & 7 & 2 & 8 & 0 \\ & & & & & 7 & 4 & 0 & 7 & 3 & 6 \\ & & & & 8 & 6 & 4 & 1 & 9 & 2 \\ & & & 9 & 8 & 7 & 6 & 4 & 8 \\ + & 1 & 1 & 1 & 1 & 1 & 0 & 4 \\ \hline = & 1 & 2 & 1 & 9 & 3 & 1 & 8 & 1 & 2 & 2 & 2 & 4 \end{array}$$

Runtime: $O(n^2)$

n numbers each $O(n)$-long

Can recursion help? Want: Smaller subproblems that if solved, make progress

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.

$$\underbrace{123}_{a_1}\underbrace{456}_{a_0} \qquad n=6$$

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

By "FOIL"

$$ab = a_1 b_1 \cdot 10^n + (a_1 b_0 + a_0 b_1) \cdot 10^{n/2} + a_0 b_0$$

4 $\frac{n}{2}$-digit multiplications

+ Carries & bitshifts $O(n)$

$$\underbrace{T(n)}_{\text{runtime on input length } n} = 4T\left(\tfrac{n}{2}\right) + O(n)$$

$$= O(n^2)$$

Idea 2: Karatsuba recursion

We'll see why next time!

Rewrite $(a_1 b_0 + a_0 b_1)$

$$= \underbrace{(a_1 + a_0)(b_1 + b_0)}_{\text{One more } \frac{n}{2}\text{-digit multiply}} - a_1 b_1 - a_0 b_0$$

precomputed

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

$$= O(n^{1.59})$$