# CS 331, Fall 2025
# Lecture 1 (8/25)

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_fa25.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
Typos, confusing examples

2) Ed

– Announcements, practice tests
– Ask questions about lecture, HW

3) Canvas

- HW turn-in, grades released
- Lecture videos

Save notes

Grade breakdown

40% HW (6x, lowest dropped)

30% Midterms (2x)

30% Final exam

- No AI on HW! Best opportunity to practice for tests.
  Acceptable use: asking to explain lecture notes, examples
- Coding component
  ≤1 coding question / HW, Python w/ starter code
  Let me know early if concerns about background.
- Tip: ask questions early, use OH + discussions
  (Ed, feedback forms)


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

# Background: Induction (Part I, Section 2)

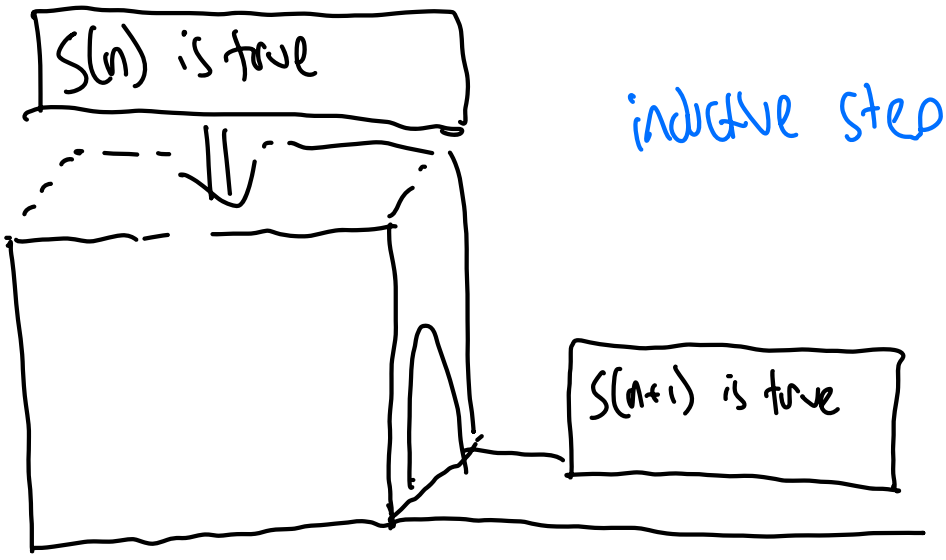
You want to prove a statement $S(n)$ for all $n \in \mathbb{N}$.

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

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

① $S(1)$ is true — base case

② $S(n)$ is true ⇓ $S(n+1)$ is true — 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 $n+1$ $2^n$

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

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

$$1 + x + \dots + x^{n-1}$$

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

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

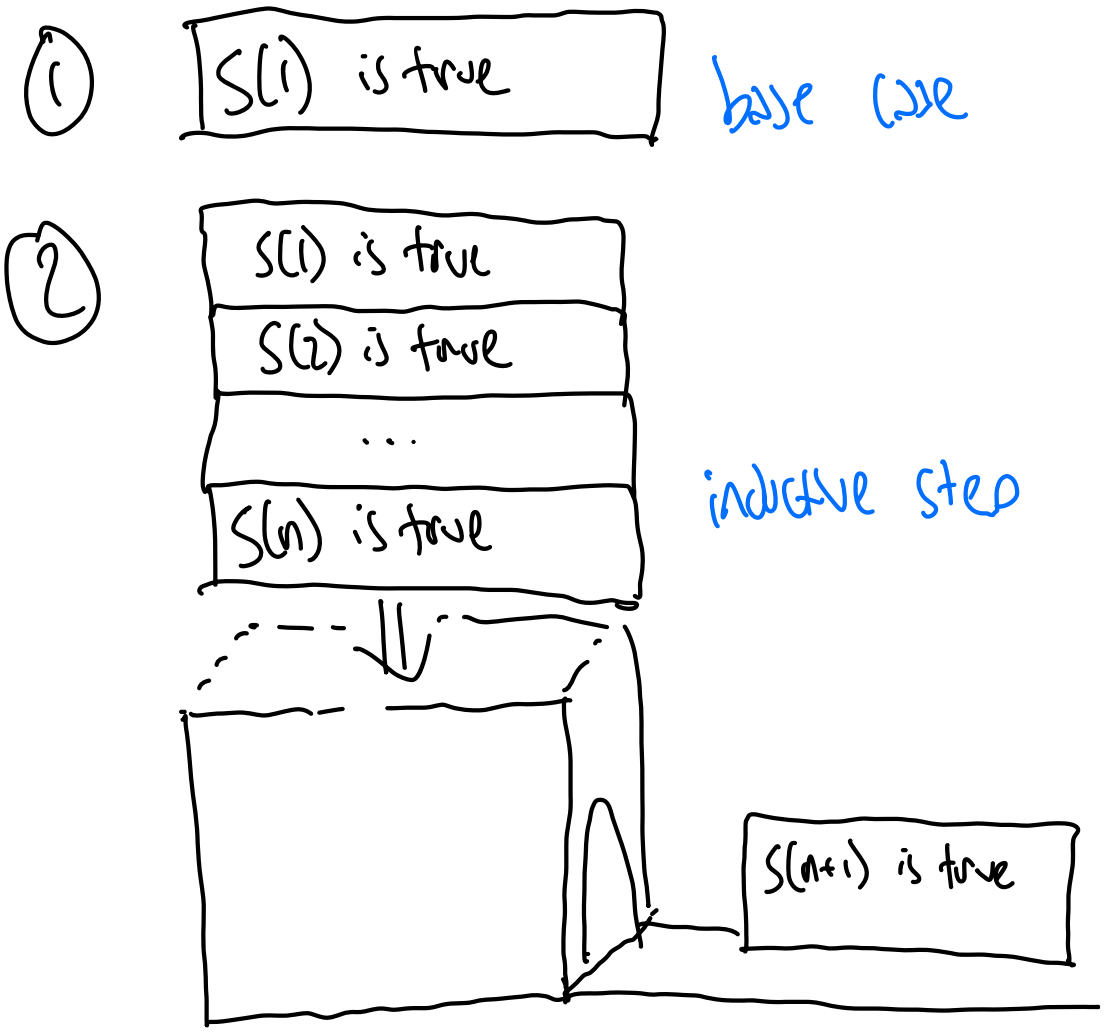
$$= (x-1)\left(x^n + \sum_{i=0}^{n-1} x^i\right)$$ ✓ ∎

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

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

Sum of geometric series.

Another version: Strong induction.




We get this stronger assumption for free during normal induction!

Still proves all S(n).

**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 $n = \underbrace{a}_{\text{Strong induct: } p_1 p_2 \cdots p_i} \cdot \underbrace{b}_{q_1 q_2 \cdots q_j}$ ✓

# Background: Asymptotics (Part I, Section 3)

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

You can use as many steps as you want.

In algos we care about efficiency along with correctness

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

**Philosophy** If $n$ is small, Algo 1, 2 both fast

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

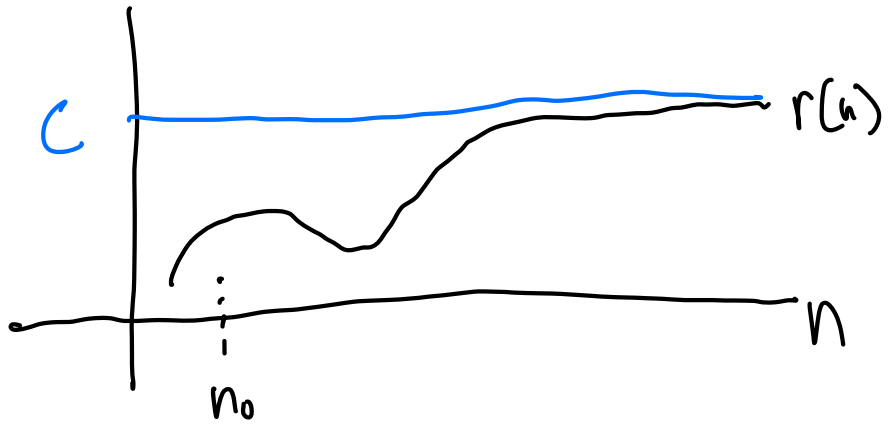
"asymptotics"

The zoo:

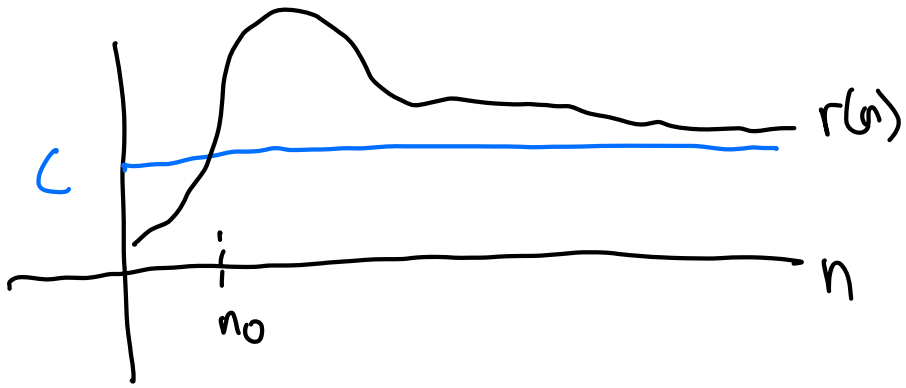
$$f(n) = \begin{cases} O(g(n)) \\ \Omega(g(n)) \\ \Theta(g(n)) \\ o(g(n)) \\ \omega(g(n)) \end{cases} \iff \frac{f(n)}{g(n)} = \begin{cases} O(1) \\ \Omega(1) \\ \Theta(1) \\ o(1) \\ \omega(1) \end{cases}$$

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

... $O(1)$? $\forall n \geq n_0,\ r(n) \leq C$ Constant independent of $n$, like 100



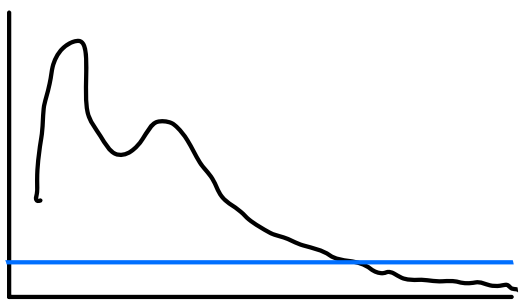

... $\Omega(1)$? $\forall n \geq n_0,\ r(n) \geq C$




... $\Theta(1)$? 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 $(2^{100n}, e^n, \dots)$

Polynomial $(n^6, 12n^3, n^2+3, \dots)$

Polylogarithmic $(\log^3 n, 3\log^4 n, \log n + 2, \dots)$

Helpful rule

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

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

**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: Very useful 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)

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

**Stack/Queue** Based on LinkedList

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

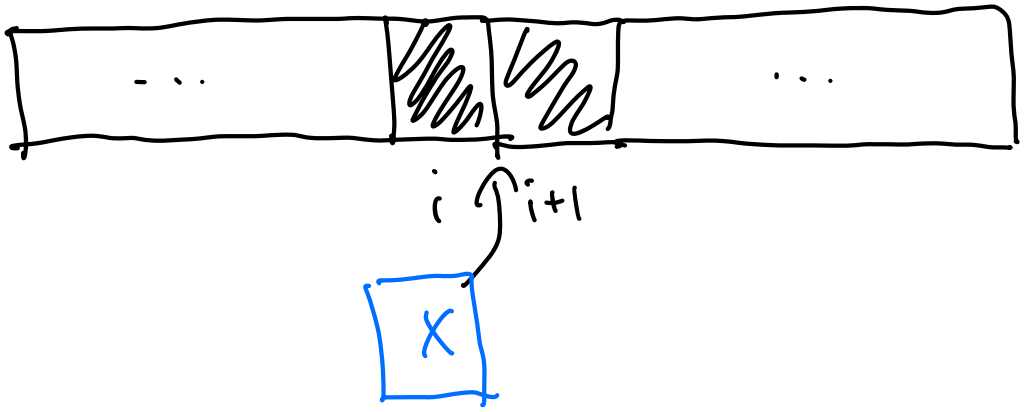
**BST**
- Insert O(log(n))
- Delete O(log(n))
- Query O(log(n))
- Search O(log(n))

**HashTable** not used until Part VII
- Init O(1)
- Insert O(1) (only in expectation ...)
- Search O(1)
- Delete O(1)

**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 $\mathcal{T}$ task for many ($>> n$) values of $i$, possibly different

# Orders of magnitude (Part II, Section 4)

Taxonomy of runtimes:

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

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

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 $n \leq 10^7$

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

**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: $O(n)$

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

grade-school also: each output takes $O(1)$

How about multiplication?

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

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. $123456$ ($a_1$ = 123, $a_0$ = 456), $n = 6$

$987654$ ($b_1$ = 987, $b_0$ = 654)

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 $O(n)$

$T(n)$ = runtime on input length $n$

$T(n) = 4T(\frac{n}{2}) + O(n)$

$= O(n^2)$

Idea 2: Karatsuba recursion

well see why next time!

Rewrite $(a_1 b_0 + a_0 b_1)$

$= (a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0$

$(a_1 + a_0)(b_1 + b_0)$: one more $\frac{n}{2}$-digit multiply

$a_1 b_1$, $a_0 b_0$: precomputed

$T(n) = 3T(\frac{n}{2}) + O(n)$

$= O(n^{1.59})$