CMSC 27200 - Guerilla Section 1

Asymptotics, recursion, and induction.

Take a picture for the kids.

Topics.

1. Some words about succeeding in the course.
2. Big-O, Big-Ω, Big-Θ.
    - Definitions.
    - In the context of analysing algorithms.
    - Worst-case, best-case?
    - The limit "trick".
3. ~~Recursion, divide and conquer~~, proofs by induction.
    - ~~What is recursion?~~
    - ~~What are recurrence relations?~~
    - ~~What is divide and conquer?~~
    - ~~Proofs by induction when~~ proving correctness.

②

Words about the course.

→ What is an algorithm?

- Precise sequence of instructions to compute a task.
- ↳ so that we can analyze it w/ mathematical precision.

- You use algorithms everyday.
  - What did you do when you woke up this morning?
    - Get out of bed
    - Get dressed.
    - Took a shower.
    - Brushed teeth
    - Ate breakfast.

- Algorithms are used outside of computer science.
  - AP USH DBQs — Industrial revolution happened because the Ford model T was constructed

on an assembly line.

- Construct chassis → attach wheel → install seats ... etc.
- Algorithm for building a Ford model T.

Main point: The study of algorithms is the study of efficiently solving problems.

→ Stress this ...

→ This course: you'll learn different ~~para~~ algorithmic paradigms ~~and toolkits~~ to solve different classes of problems.

- You'll also learn a toolkit to analyze ~~these~~ algorithms ~~for~~ rigorously
- Focus on two things ① correctness, ② efficiency.

⟹ Some words about succeeding.

This course runs about 9 weeks and covers quite a bit of ~~content. It's not easy by any means~~. challenging content. The course is not easy by any means.

I usually give two pieces of advice, really more statements...

~~1 Don't get lost~~

~~2 Don't feel alone.~~

1 You're not lost.

2 You're not alone.

3 You can do it.

Okay what do I mean...

⟹ You're not lost.

The feeling of "I have no idea wtf is going on" comes from a loss of context i.e...

- I don't know <u>why</u> these words are being said.
- I don't know <u>why</u> I'm reading this.

When you lose context it's helpful to backtrack.

⟶ What was the last thing I understood?

⟶ What was the next thing I wanted to understand?

→ How did I <u>get stuck</u>?

Then you take the answer to those three questions and present it to a TA.

↪ "You're not lost" you're just trying to figure out where you ended up.

⟹ You're not alone

Okay so like the most common comment I get from a student is "I feel so behind <u>everyone else</u>"

⑥

Idk man look around you. Ostensibly you're here because you recognize this course is difficult and you want to be proactive about your learning

→ that means everyone here has had this feeling. "I'm so far behind everyone else".

→ you're not alone, there are others struggling just as you are.

↳ reach out: ask a TA / Lorenzo for help. post on Ed, send an email

↳ Turn around, introduce yourself: make a new friend. find a large study group.

- At most 2 collaborators on a HW question
- But as many folks you want to discuss non-hw stuff.

↳ Be kind: when you understand something.

explain it to others.

↳ write a note, share it on ed.

→ help ~~others~~ the folks around you.

⟹ You can do it.

~~We try our best~~ Our job is to make sure you succeed in this course.

- We love the subject matter and we want to share it in a way that you understand,
- We try our best to give you ~~the~~ sufficient resources. because you can succeed.
- Y'all are smart, y'all work hard, just try your best and be proactive about your learning.

Asymptotics.

These are tools for understanding how fns scale relative to one another.

→ For all fns we'll deal w/ today.

$$f : \cancel{\mathbb{N}} \to \mathbb{R}$$

$$\mathbb{Z}_{>0}. \longrightarrow \{1, 2, \dots +\infty\}.$$

Now there are three pieces to remember.

→ Big-O.

Defn: Given two functions $f, g : \mathbb{Z}_{>0} \to \mathbb{R}$, we say

$$f = O(g)$$

if $\exists\ c > 0,\ n_0 > 0$ s.t. $\forall\ n \geq n_0$

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

→ Big-Ω

Defn: Given two fns $f, g : \mathbb{Z}_{>0} \to \mathbb{R}$, we say

$$f = \Omega(g)$$

if $\exists\, c>0,\ n_0>0$ s.t. $\forall n \geq n_0$.

$$f(n) \geq c \cdot g(n).$$

$\longrightarrow$ Big-Θ.

<u>Defn</u>: Given two functions $f, g : \mathbb{Z}_{>0} \longrightarrow \mathbb{R}$ we say

$$f = \Theta(g)$$

if $f = O(g)$ <u>and</u> $f = \Omega(g)$.

$\Longrightarrow$ How do you think about these things...

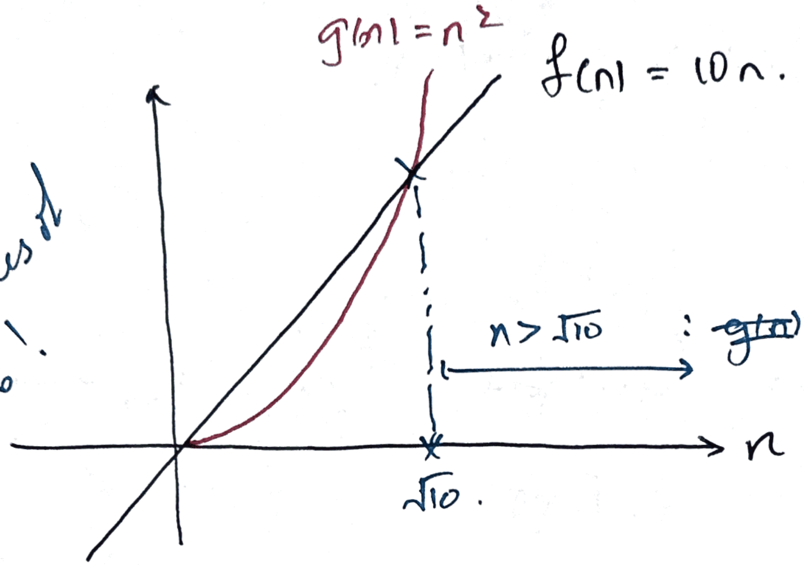
1) ~~Landau This notation (called Landau notation) captures "classes of fns which all scale the same as $n \to \infty$"~~

$$\cancel{\Theta(g) = \{ f : \mathbb{Z}_{>0} \to \mathbb{R} : f = O(g) \}.}$$

Take an <u>example</u> $f(n) = 10n$, $g(n) = n^2$. and focus on big-O

(10)

1) To say that $f = O(g)$ is like saying
"g scales ~~as~~ at least as fast as f as $n \to \infty$"



there could be many choices of c, n_0!

$\exists$ $c > 0$, $n_0 > 0$ s.t. $\forall n \geq n_0$, we have

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

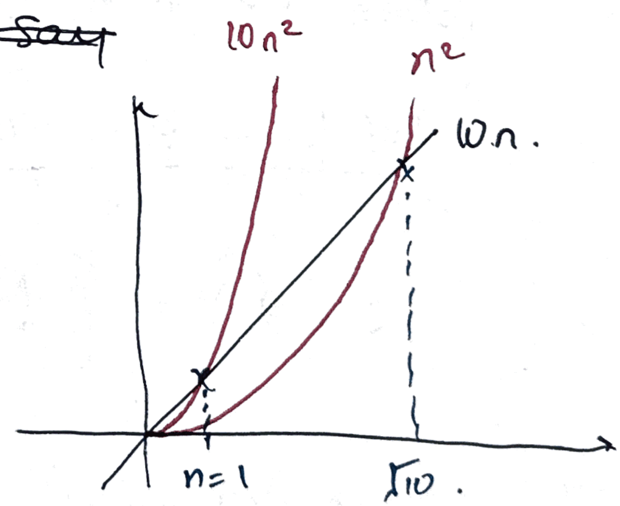
$\longrightarrow$ pick $n_0 = \sqrt{10}$, $c = 1$
then $\forall n \geq \sqrt{10}$ we have

$$f(n) = 10n \leq n^2 = c \cdot g(n).$$

$$\longrightarrow f = O(g).$$

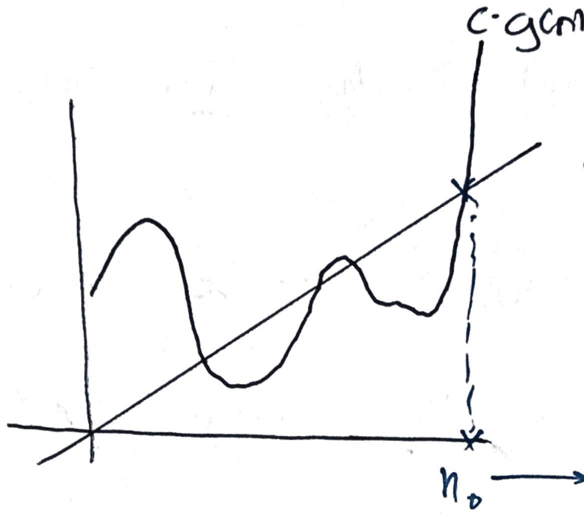
~~2) Alternatively you can say~~

Alternatively, let $n_0 = 1$ and $c = 10$. Then we can show $\forall n \geq n_0 = 1$



$$f(n) = 10n \leq 10n^2 = c \cdot g(n)$$

2) Alternatively you can say g(n) <u>asymptotically</u> <u>upperbounds</u> f(n)



If $f = O(g)$ then
$\forall n \geq n_0$,
$f(n) \leq c \cdot g(n)$
<u>has</u> to hold.

3) Finally you can think of $O(g(n))$ as denoting a <u>class of fns</u> which <u>all</u> have g(n) as an <u>asymtotic upperbound</u>

f(n) { 10n, 5n + 3, n·log(n), 17 } → $O(g(n))$ for $g(n) = n^2$.

g(n) is your representative of <u>this class</u>

(12)

→ What about Ω and Θ?

$\Omega \rightarrow$ $f(n) = \Omega(g(n))$ can be translated to "$g(n)$ <u>asymptotically</u> <u>lowerbounds</u> $f(n)$."

$\Theta \rightarrow$ $f(n) = \Theta(g(n))$ can be translated to "$g(n)$ <u>asymptotically</u> <u>upper bounds</u> and <u>lower bounds</u> $f(n)$ ~~s~~"

But actually take apart that defn

$f(n) = O(g(n)) \longrightarrow \exists\ c_1 > 0,\ n_1 > 0$ s.t.

$\forall\ n \geq n_1$

$f(n) \leq c_1 g(n)$

$f(n) = \Omega(g(n)) \longrightarrow \exists\ c_2 > 0,\ n_2 > 0$ s.t.

$\forall\ n \geq n_2$

$f(n) \geq c_2 \cdot g(n)$.

This means if $n_0 = \max(n_1, n_2)$ then.

$\forall n \geq n_0$

$$c_2 g(n) \leq f(n) \leq c_1 g(n).$$

→ Up to $\$$ constants, $f(n)$ scales the same as $g(n)$.

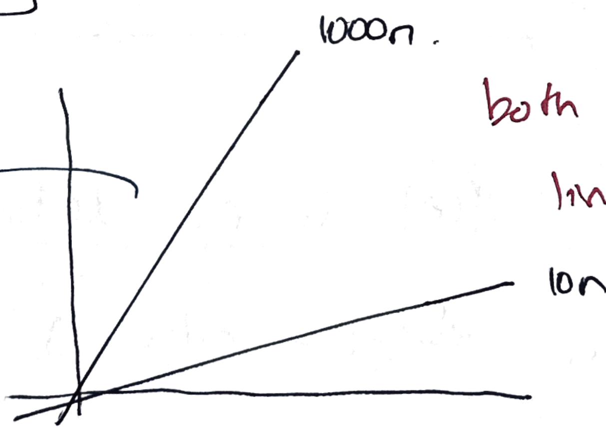
→ Questions?

⇒ Hazards.

1) How can $1000n = O(10n)$?

Like certainly $1000n \geq 10 \cdot n$ for all n.

→ This is how big * is a notation capturing scale.



baked into the defn. you get to pick this c.

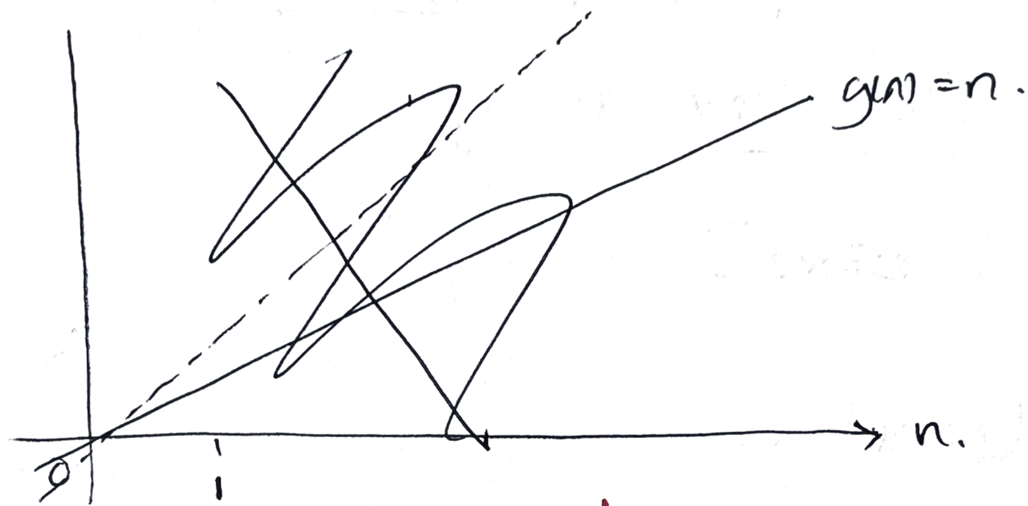
both of these scale linearly so you would intuitively want that $1000n = O(10n)$

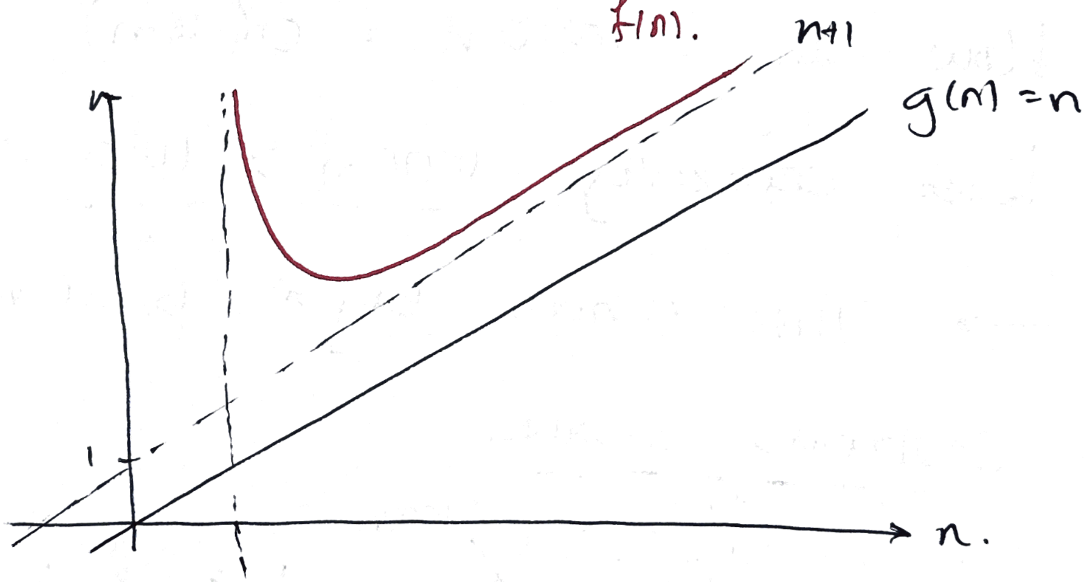
$c = 100$ gives $\forall n \geq 1$ $f(n) = 1000n \leq 1000n = 100 \cdot g(n)$

(in fact $1000n = \Theta(10n)$).

2) Being careful about how you quantify functions

Imagine $f(n) = \dfrac{n^2-1}{n-1}$ $\quad g(n) = n.$





- At $\underline{n=1}$, $f(n)$ is <u>not defined</u>! Technically that means $f(n) = O(g(n))$ is an <u>ill-defined statement</u>. To be precise we would need to either change our <u>definition</u>

to quantify over $f: \mathbb{Z}_{>0} \to \mathbb{R} \cup \{+\infty\}$ or "fix the function". i.e. define

$$f(n) = \begin{cases} 0 & \text{if } n = 1 \\ \frac{n^2-1}{n-1} & \forall\, n \geq 1. \end{cases}$$

Then we could prove that $f(n) = O(g(n))$.

→ ~~why does this~~ Be careful about what class of fns you define this notation for.

→ Why does this matter? Because there's a hw question which asks to show that this defn. is equivalent to another.

Def 1 (DPV).

~~$f(n), g(n)$~~

Suppose $f, g : \mathbb{Z}_{>0} \to \mathbb{R}_{>0}$ then $f(n) = O(g(n))$ iff $\exists\, c > 0$ s.t. $\forall n \geq 1$ $f(n) \leq c \cdot g(n)$.

16

<u>def 2</u> (KT): $f, g : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{\geq 0}$ we say

$f = O(g)$ iff. $\exists c > 0, n_0 > 0$ s.t. $\forall n \geq n_0$

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

3) Best-case, worst-case ... average case?

There is this <u>misconception</u> that ...

$\Omega \longrightarrow$ best-case analysis.

$O \longrightarrow$ worst-case analysis.

$\Theta \longrightarrow$ ... average-case?

Here it's important to distinguish what

the <u>notation</u> ~~precisely means~~ <u>formally</u>

<u>means</u> and what we <u>use it to express</u>

<u>qualitatively</u>.

For example the following statements are all true

1) The best-case running time for insertion sort over all inputs $\Theta(n)$.

2) → It's $O(n)$ and $\Omega(n)$.

→ best-case is defined as running insertion sort over sorted lists.

→ when you quantify ~~over the running time over~~ over running insertion sort on sorted lists.

↳ the running time as a fn of the size of the list is always

~~c.n~~. $T(n) = c.n$.

2) The running time of insertion sort quantified over any input is $\Omega(n)$ and $O(n^2)$.

→ running time analyses are quantified over what class of inputs you chose to focus on.

↳ Once you fix the set of inputs, then you use big-O, Ω, Θ to quantify the running time.

* Big-O, Ω, Θ has nothing intrinsically to do w/ what ~~class of inputs you quantify~~ regime you're analyzing your algorithm in. It's just ~~a set of~~ mathematical language to express asymptotic upper / lower bds.

→ Remark on how we use big-O, Ω, Θ.

How do we use this notation?

- We ask the question: ~~how does the running time~~ For a set of inputs (maybe all possible), how does the running time of the algorithm (roughly the # of ~~instr~~ "atomic" instructions) scale as a fn of the input size?

- "Class of inputs"
- "Atomic instruction"
- "Input size".

HW question to show that "how you model things" could give you diff. running times

→ matrix multiplication.

"Class of inputs" All $n \times n$ real matrices.

"Atomic instructions"

$O(1)$ time to do $+, *, \div, -$.

↓

"Input size"

$n :=$ dimension of matrix.

"Atomic instructions"

$O(b)$ to do $+, -$

$O(b^2)$ to do $*, \div$

↓

"Input size"

$n :=$ dimension matrix

$b :=$ max bit length.

→ Importance of scale

- The reason why we care about scale. ~~is because~~ and not constants / lower order terms is because then it gives us a sense of how $\underset{\text{fast}}{\text{a}}$ the algorithm runs in a machine independent way.

- Two instruction sets could implement multiplication in like

  b^2 vs. $10b^2$ instructions.

- Now if you compare two algorithms that run in say... $O(n^2 \cdot b^2)$ vs. ~~$O(n \log n b^2)$~~ $O(n^2 + b^2)$ time. then you know the speed up is not just because one machine is faster.

→ Stress how important of a difference that actually is in practice

↳ Theoretically faster algos will always beat improvements in hardware

↳ Questions?

Limits and Asymptotic Notation

How do you prove that $f(n) = O(g(n))$.

→ By definition: Find $c > 0$, $n_0 > 0$ s.t.

$$\underbrace{\forall n \geq n_0 \quad f(n) \leq c \cdot g(n)}_{\text{Cumbersome}}$$

→ Using limits:

Fact: If $\lim\limits_{n \to \infty} \frac{f(n)}{g(n)}$ exists then

1) $\frac{f(n)}{g(n)} < +\infty \Longrightarrow f(n) \leq O(g(n))$

2) $\frac{f(n)}{g(n)} > 0 \Longrightarrow f(n) = \Omega(g(n))$.

Kinda makes sense...

→ Think of $f(n) = O(g(n))$. Then "the rate at which $f(n)$ grows asymptotically is <u>at most</u> $g(n)$." : Two cases.

↳ $g(n)$ asymptotically faster...

$$\frac{f(n)}{g(n)} \longrightarrow 0.$$

↳ $g(n)$ asymptotically the same as $f(n)$

$$\frac{f(n)}{g(n)} \longrightarrow c \quad \text{some finite constant} > 0.$$

<u>Examples</u>:

1) Is ~~$10n = O(n^2)$? Yeah~~... $10n + 3 = O(n)$?

$$\lim_{n\to\infty} \frac{10n+3}{n} = \lim_{n\to\infty} 10 + \frac{3}{n} = 10. \checkmark$$

2) Is $2^n = O(3^n)$? yeah...

$$\lim_{n\to\infty} \frac{2^n}{3^n} = \lim_{n\to\infty} \left(\frac{2}{3}\right)^n = 0.$$

3) fix any $\underline{\epsilon > 0}$, is $n^\epsilon = O(\log n)$?

$$\lim_{n\to\infty} \frac{n^\epsilon}{\log n}.$$

L'hopital!

If f and g are differentiable and $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ is an indeterm form then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

$$= \lim_{n\to\infty} \frac{\epsilon \cdot n^{\epsilon - 1}}{1/n}.$$

$$= \lim_{n\to\infty} \epsilon \cdot n^\epsilon$$

$$= +\infty \;!$$

$\longrightarrow$ And so in fact $n^\epsilon \geq \Omega(\log n)$ $\forall \epsilon > 0$.

$\longrightarrow$ Questions?

24

Recursion

A function ~~f : S →~~ f is *recursive* if it's defined in *terms of itself*. The defn of a recursive fn is called a *recurrence relation*

→ Anatomy

1) *Base case*: fn evaluated on fixed input.

2) *Inductive case*: fn evaluated on itself.

Example: factorial.

From *CMSC 27100*: $n! = 1 \cdot 2 \cdot \ldots \cdot n$.

We can define this *recursively*.

$$\underbrace{T(n) = \begin{cases} 1 & \text{if } n = 0, 1. \\ n \cdot T(n-1) & \text{if } n > 1. \end{cases}}_{\text{recurrence relation}}$$

~~And now~~.

"recursive fn on the natural #s"