L27: Recurrence Relations
&
Introduction to Big-O

Announcements

Upcoming Dates

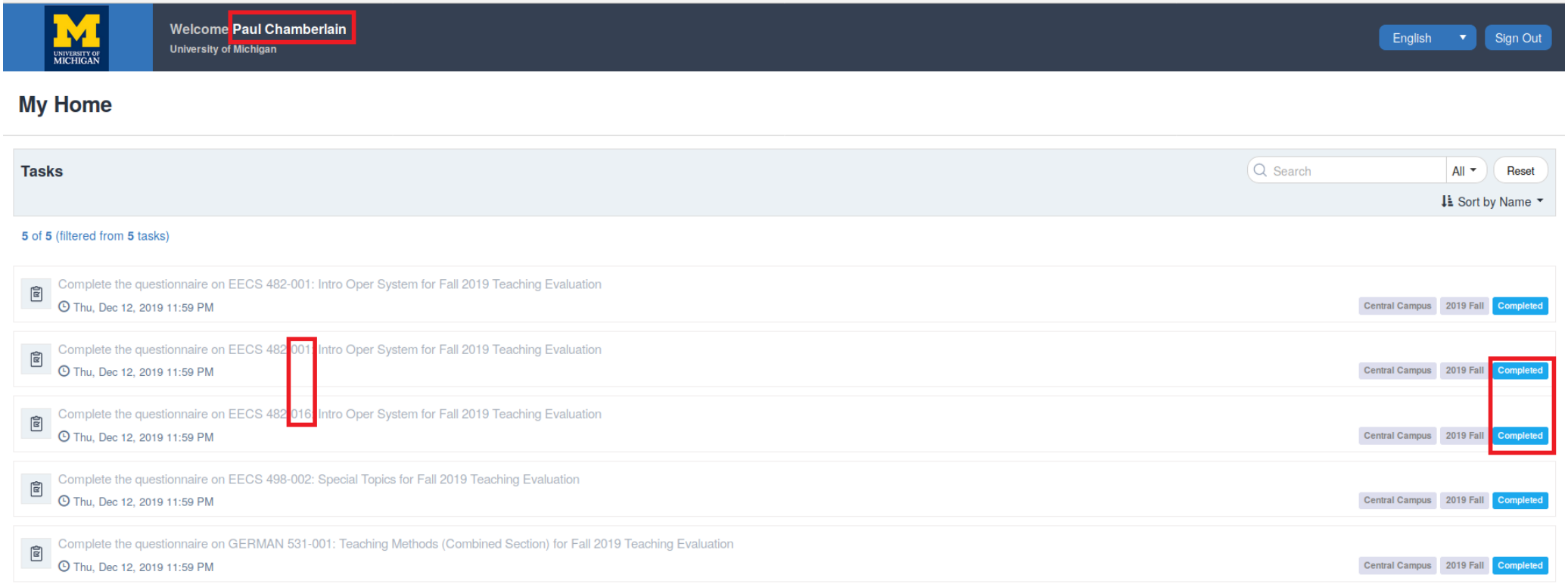
- **Tomorrow: IA application deadline**
- **Tues 4/19 (last day of classes)**
  - **HW 10 due**
- **Wed 4/20: Surveys due**
- **Tues 4/26: Exam 3**

- **EECS 203 Stickers!!!**
  - **Available in lecture on Tues**
  - **Also at other times TBD**
- **Office Hours will continue after classes end**
  - **Likely with a different schedule**
  - **Check the OH calendar for the most up-to-date info**

No Prof. Diaz Office Hours today.
Rescheduled for tomorrow.

Extra Credit Course Evaluation Due 4/20

- Submit a screenshot including your name, and the label "Completed" for both the lecture and discussion section



Exam 3 Planning

	Sunday	Mon	Tuesday	Weds.	Thursday	Friday	Saturday
This week				Practice Exams released	Today		
4/17-4/23			*Last day of classes* HW 10 due Practice Exam Solutions released	Surveys due Special Topics Review #1 4-6pm	Special Topics Review #2 4-6pm		Review session #1: 1-4 pm
4/24-4/30	Review session #2: 1-4 pm		**4/26: Exam 2: 7-9pm**				

- Exam 3 covers: Graphs, Counting, Probability, Meta-Counting (HW 8 – 11)
- Please note: the material is *inherently cumulative*
  - (e.g., you could be asked to count bijections, or prove something about graphs/counting/probability/etc
- See "Exam 3 Information" doc on Canvas *(coming soon!)* and related Piazza post

Learning Objectives

After today's lecture (and associated readings, discussion & homework), you should be able to:

- State what a recurrence relation is and how it is useful in computer science
- Given an English description of a recurrence, determine the associated mathematical recurrence, including initial conditions
- Define big-O, big-Θ, and big-Ω, mathematically and in plain English
- know the relative growth rates of the standard functions used to characterize complexity
- Determine the runtime of mathematical functions
  - For defined functions
  - For combinations of functions, given their individual runtimes

Outline

- **Recurrence and Recurrence Relations**
  - **Ex: Tower of Hanoi**
  - **Ex: Climbing stairs**
  - **Ex: Katie's Pandemic Outfits**

- Big-O Notation
  - Introduction & Definitions (Big-O, Big-Omega, Big-Theta)
  - Examples: Runtime of mathematical functions
  - Properties for Combining Functions

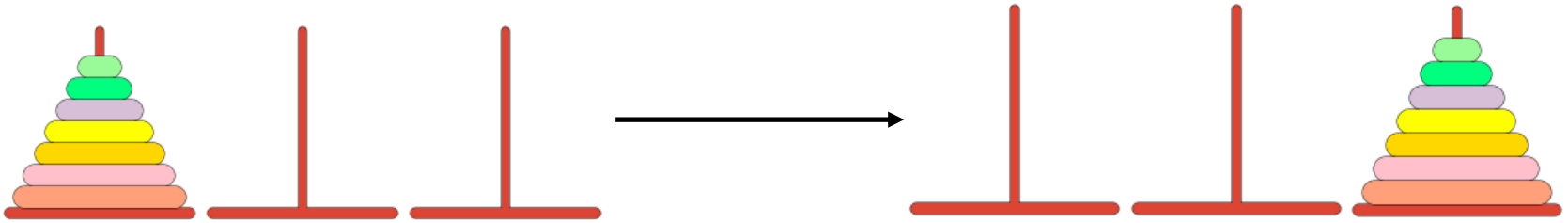
- Next time: Big-O and recurrence relations come together

Recursion

- Recursion shows up in many places
  - ***Programs*** that call themselves
  - ***Sequences***: each term defined by previous terms
  - ***Sets***: elements in the set determined by others in the set
  - ***Physical processes***: state at time *t* depends on the state of the process at prior times

Example: Tower of Hanoi

- Recall: Tower of Hanoi rules of the game.
  - ***n*** discs initially on peg 1.
  - Can only move one disc at a time.
  - A disc can only be put on a larger disc or an empty peg.
  - GOAL: move all discs from peg 1 to peg 3.



Start Finish

- We previously proved that ToH with n disks has a solution $\forall n \geq 0$
  - (Hooray for induction!)

- Next: Count the **number of moves** required to solve the Tower of Hanoi puzzle for n disks.

Example: Tower of Hanoi

Count the number of moves required to solve the Tower of Hanoi puzzle for *n* disks.

- Set-up: Let $T(n)$ be the number of moves required for *n* disks.
- Brute-force approach
  - For a given *n*, enumerate the moves starting with the smallest disk and continue until you're done.

Comment: Though exhaustive, this approach is exhausting!

Revisit Inductive Proof for ToH

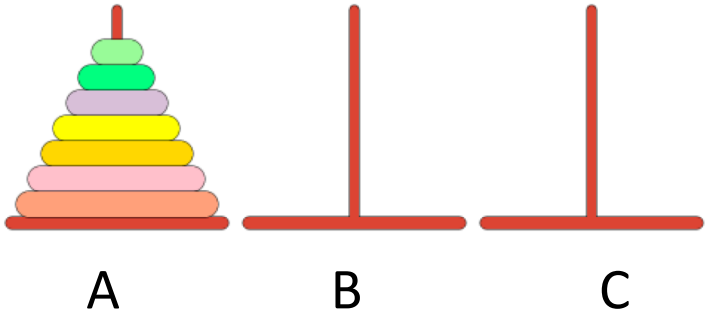
- **For all $n \in \mathbb{N}$, $P(n)$:** The Tower of Hanoi puzzle has a solution with n discs initially on peg A.

- **Proof:** Base Case : P(0)
  When n = **0**, start configuration **=** end configuration

Inductive Step:

Assume P(k): the puzzle has a solution for k discs.

Want to show P(k+1): the puzzle has a solution for $k+1$ discs.



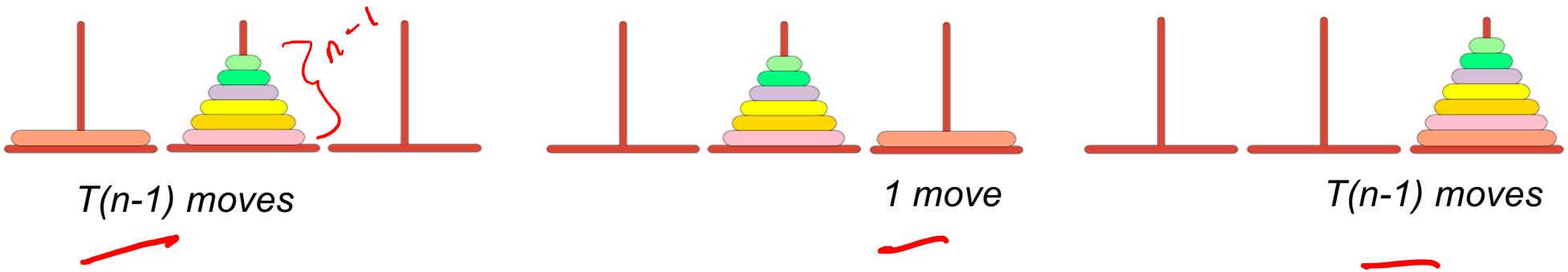
1. Use **inductive hypothesis** to move top k-1 disks to peg B.
2. Move biggest disc to peg C.
3. Use **inductive hypothesis** to move top k-1 disks to peg B.

This proof can serve as a guide to help us count the number of moves required to solve the puzzle for n disks.

Example: Tower of Hanoi

Count the number of moves required to solve the Tower of Hanoi puzzle for *n* discs. T(n)

- Recursive approach



- Define T(n) = number of moves when there are n disks.
  - T(n) = T(n-1) + 1 + T(n-1)
    = 2T(n-1) + 1
  - Initial condition/base case: T(0)=0

Recursive Algorithm for ToH in Pseudocode

ToH(n,A,B,C) : *move n disks from A to C using B if necessary.*

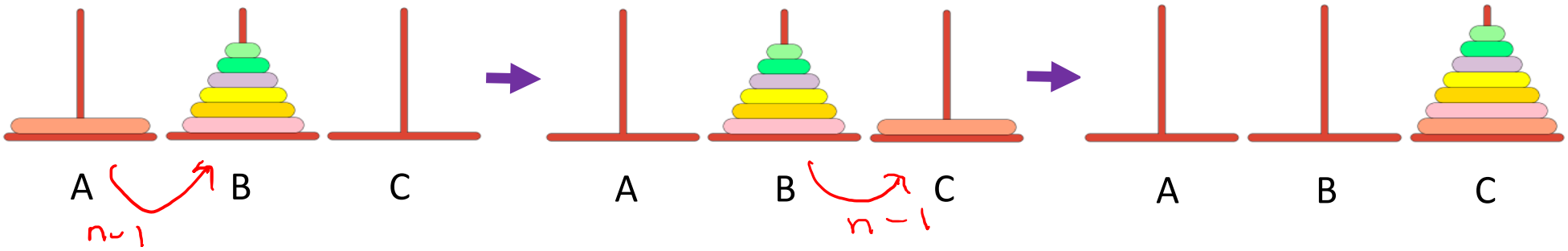
If n=0, stop → Base case

ToH(n-1,A,C,B) → T(n-1) moves

Move disk n from A to C → 1 move

ToH(n-1,B,A,C) → T(n-1) moves

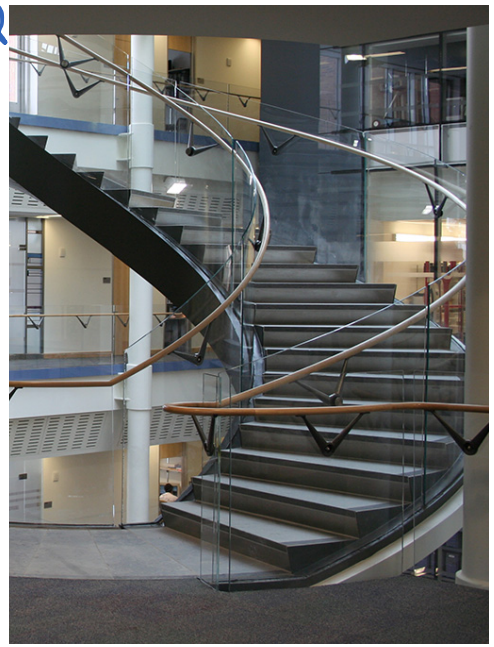
Recursive approach



- Define T(n) = number of moves when there are n disks.
  - T(n) = T(n-1) + 1 + T(n-1)
    = 2T(n-1) + 1
  - Initial condition/base case: T(0)=0

Example: Climbing Stairs

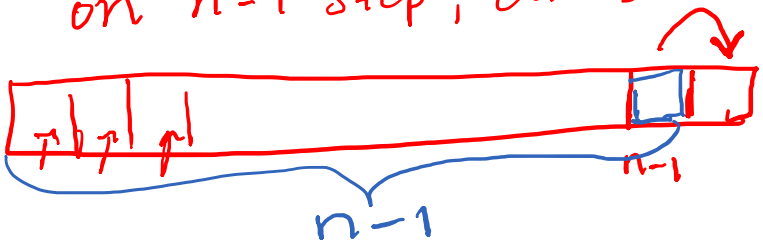
How many ways are there to climb n stairs if you can take one stair or two stairs at a time?



$T(n)$: # ways to climb n stairs

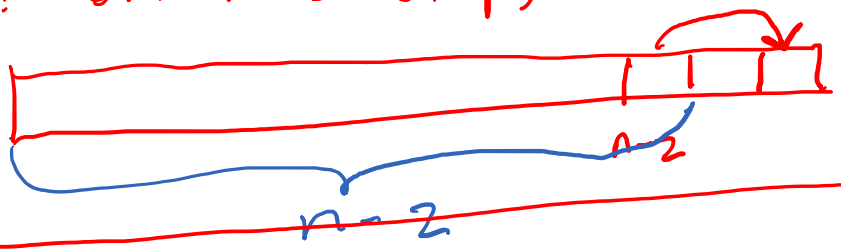
set up cases based on where you were immediately previous to the n^{th} step

Case 1: on $n-1$ step, then climb one step



$\Rightarrow T(n-1)$ ways

Case 2: on $n-2$ step, then climbed 2 steps



$\Rightarrow T(n-2)$ ways.

$$T(n) = T(n-1) + T(n-2)$$

initial conditions: $T(0) = 1$ $T(1) = 1$

#ways to do nothing = 1

Example: Climbing Stairs

Count the number of ways to climb *n* stairs if you can take one stair or two stairs at a time.

- Set-up: Let $s(n)$ be the number of ways to climb *n* stairs
- Avoid brute force, find a recurrence relation
- Consider the different ways you can arrive at the n-th stair
  - *Climb to the n-1st stair then climb one stair*

  or
  - *Climb to the n-2nd stair then climb two stairs at once*.
- There are $s(n-1)$ ways to do the first option and $s(n-2)$ ways to do the second option

$$s(n) = s(n-1) + s(n-2)$$

Need 2 initial conditions since recurrence goes back to $n-2$:

$$s(0) = 1 \text{ and } s(1) = 1$$

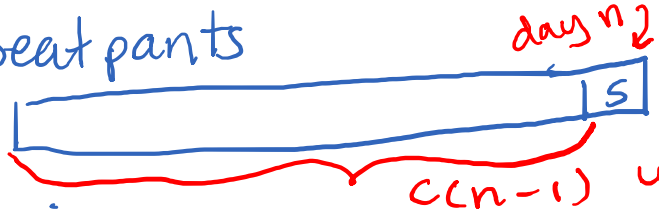
Example: Katie's Pandemic Outfits

Katie has three outfit choices while social distancing. Each day she wears either pajamas, sweatpants, or jeans. Her only rule is that she never wears jeans on two or more days in a row.

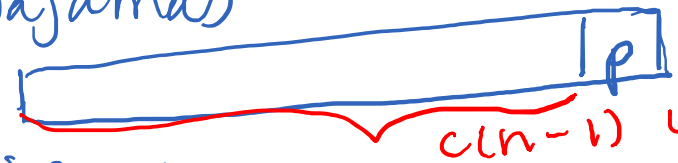
Let $c(n)$ represent the number of ways Katie can choose outfits across n days, $n \geq 0$. Find a recurrence relation for $c(n)$ including the initial conditions

Cases based on what she wore on day n.

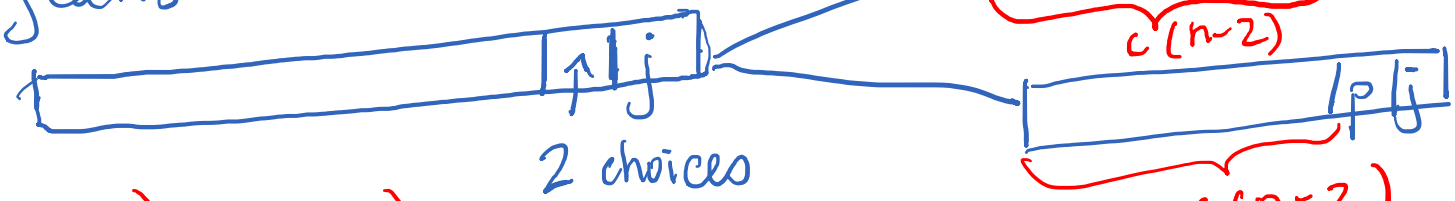
Case 1: sweatpants



case 2: pajamas



case 3: jeans



$c(n) = c(n-1) + c(n-1) + 2c(n-2)$

$c(n) = 2c(n-1) + 2c(n-2)$

Initial conditions:

$c(0) = 1 \quad c(1) = 3$

Example: Katie's Pandemic Outfits

Katie has three outfit choices while social distancing. Each day she wears either pajamas, sweatpants, or jeans. Her only rule is that she never wears jeans on two or more days in a row.

Let $\boldsymbol{c(n)}$ represent the number of ways Katie can choose outfits across $\boldsymbol{n}$ days, $n \geq 0$. Find a recurrence relation for $\boldsymbol{c(n)}$ including the initial conditions

Solution: Consider different cases based on which pants she wears on day n

- Case 1: day n = sweatpants
  - Then for previous $n-1$ days, there were $c(n-1)$ ways
- Case 2: day n = pajamas
  - Then for previous $n-1$ days, there were $c(n-1)$ ways
- Case 3: day n = jeans
  - Then day $n-1$ was sweatpants OR pajamas (can't wear jeans 2 days in a row).
  - Case 3.1: day $n-1$ was sweatpants
    - For previous $n-2$ days, there are $c(n-2)$ ways
  - Case 3.2: day $n-1$ was pajamas
    - For previous $n-2$ days, there are $c(n-2)$ ways
- Combining all cases, we have $c(n) = 2c(n-1) + 2c(n-2)$
- Initial conditions: need 2 of them since recurrence goes back to $c(n-2)$
  - $c(0) = 1,\ c(1) = 3$

Recursion and Recurrence Relation

- Recurrence Relations
  - $T(n)$ is defined in terms of previous values, i.e., $T(k), k < n$
  - We need initial value(s) of T to fully define the recurrence
  - Recursion = induction
- So far we've seen *linear* recurrences
  - ToH: $T(n) = 2T(n-1) + 1$
  - Stair climbing: $s(n) = s(n-1) + s(n-2)$
  - Pandemic outfits: $c(n) = 2c(n-1) + c2(n-2)$
- Coming soon: Divide & Conquer recurrences
  - Take the form: $T(n) = aT\left(\frac{n}{b}\right) + cn^d$

Outline

- Recurrence and Recurrence Relations
  - Ex: Tower of Hanoi
  - Ex: Climbing stairs
  - Ex: Katie's Pandemic Outfits
- **"Big-O" Notation** aka computational complexity aka runtime
  - **Introduction & Definitions (Big-O, Big-Omega, Big-Theta)**
  - **Examples: Runtime of mathematical functions**
  - Properties for Combining Functions
- Next time: Big-O and recurrence relations come together

How do you measure how ***fast*** an algorithm is?

1st try:

An algorithm runs in time ***X(n)*** if, on input of size **n**, it always finishes within ***X(n)*** seconds.

This is **underspecified**. Which computer? Which programming language? This is a property of **an implementation** of the algorithm, not the algorithm itself.

How do you measure how ***fast*** an algorithm is?

1st try: **(bad)**

An algorithm runs in time ***X(n)*** if, on input of size **n**, it always finishes within ***X(n)*** seconds.

2nd try:

An algorithm runs in time ***X(n)*** if, on input of size **n**, it always finishes within ***X*** steps.

What is a "step?" This depends on how much detail you include in your pseudocode. Compare:

```
swap(x, y)          vs          temp := x
                                x := y
                                y := temp
```

How do you measure how ***fast*** an algorithm is?

1st try: **(bad)**

An algorithm runs in time ***X(n)*** if, on input of size **n**, it always finishes within ***X(n)*** seconds.

2nd try: **(bad)**

An algorithm runs in time ***X(n)*** if, on input of size **n**, it always finishes within ***X*** steps.

3rd try:

An algorithm runs in time ***O(X(n))*** if, on input of size **n**, it always finishes within **cX** steps, for a large enough **constant c > 0** (does not depend on **n**).

Big-O Notation

"Big-O Notation" is a way to group together **functions** of a variable with similar **growth rates**.

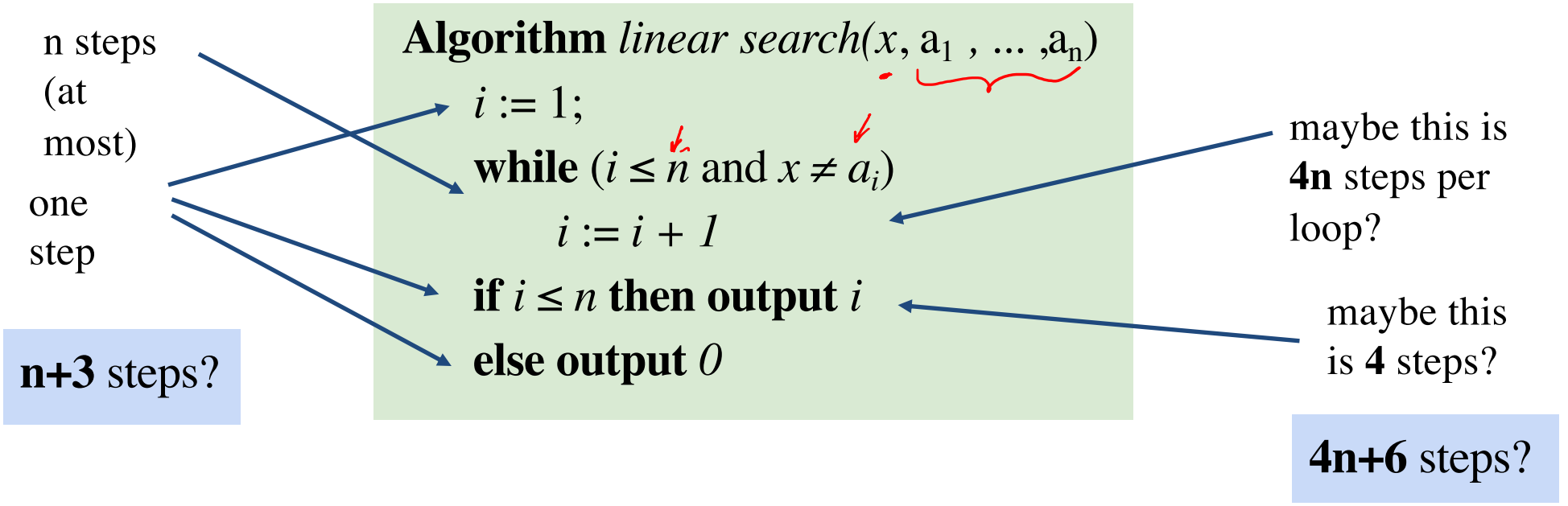
For **large enough x**:

3,
$x + \log(x)$,
$x*\log(x) + 2x - 7$,
$x^{1.372} + 3x + 1$,
$99x^2 + 98x^{1.9} + 97x^{1.8} + 96x^{1.7} + 3$

$\leq$ **100**x^2

So **all** of these functions are **O(x^2)**.

Big-O Notation



For an input list of length **n**, how many steps does this take to run?

The unambiguous answer is **O(n)** steps.

Big-O Definition

Let $f(n), g(n)$ be functions.

- Typically **f** is something complicated and annoying, and **g** is something simple and nice.

f(n) is O(g(n))

We write $f(n) = O(g(n))$
if there are constants **c**, **k** such that for all $\mathbf{n} \geq \mathbf{k}$, we have

$$f(n) \leq cg(n).$$

- "Constant" means something like 5, 7, 100 -- **not** something dependent on **n**, like $\mathbf{n^2}$.

- The following are all equivalent ways to express complexity
  - "f is $O(g)$"
  - "$f = O(g)$"
  - "$f \in O(g)$"

Most commonly used

(the "real" one)

Big-O Definition

$$f(n) = O(g(n))$$

means there are constants **c**, **k** so that for all $\mathbf{n \geq k}$, we have

$$f(n) \leq cg(n)$$

Claim: $5x^2 + 3x - 4 + 1/x = O(x^2)$

Proof:

- Let $\mathbf{c = 9}$, $\mathbf{k = 1}$. These are called **witnesses**.
- For all $\mathbf{x \geq k = 1}$, we have

$$\begin{aligned} 5x^2 + 3x - 4 + 1/x &\leq 5x^2 + 3x + 1/x \\ &\leq 5x^2 + 3x^2 + 1x^2 \\ &= 9x^2 \\ &= cx^2 \end{aligned}$$

done!

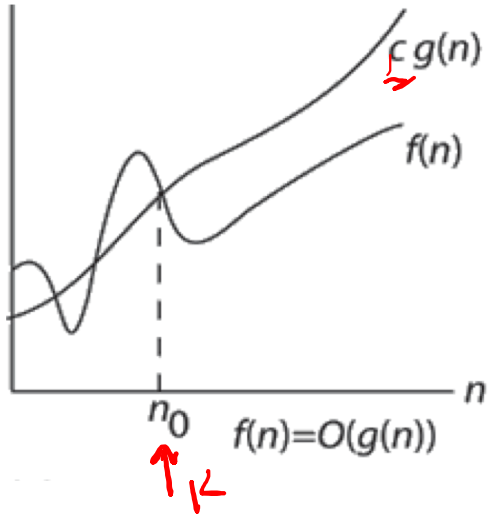
$3x \leq 3x^2$ when $x \geq k = 1$

$1/x \leq x^2$ when $x \geq k = 1$

Complexity: "Big-O", "Big-Omega", "**Big-Theta**"

- Let $f, g : \mathbb{N} \to \mathbb{R}$ (or can be $f, g: \mathbb{R} \to \mathbb{R}$)
- **Big-O**
  - "f is $O(g)$" means "f grows no faster than g"

$$\exists k, C > 0 \text{ such that } \forall n > k$$
$$|f(n)| \leq C|g(n)|$$

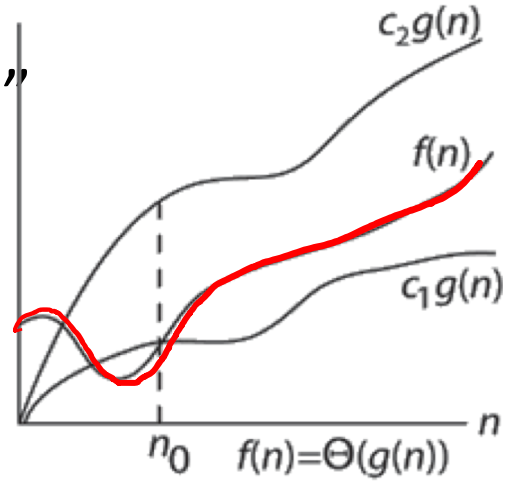


- **Big-Omega**
  - "f is $\Omega(g)$" means "f grows at least as fast as g"

$$\exists k \exists C \text{ such that } \forall n > k\text{: } |f(n)| \geq C|g(n)|$$

- ****Big-Theta****
  - "f is $\Theta(g)$" means "f grows at the same rate as g"
  - f is $\Theta(g)$ iff $f = O(g)$ and $f = \Omega(g)$

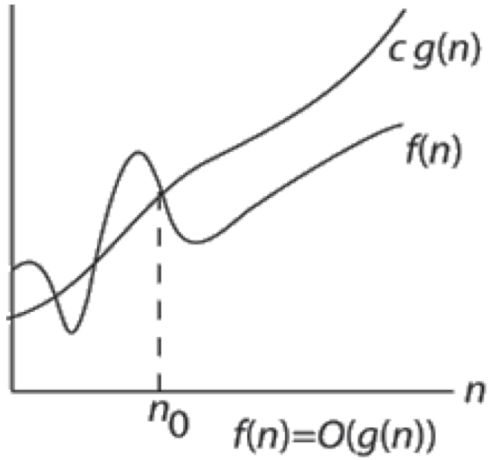
$$\exists k \exists C_1 \exists C_2 \text{ such that } \forall n > k$$
$$C_1|g(n)| \leq |f(n)| \leq C_2|g(n)|$$



Big-Oh, Big-Omega and Big-Theta

- Let $f, g : \mathbb{N} \to \mathbb{R}$ (or can be $f, g: \mathbb{R} \to \mathbb{R}$)
- **Big-O**
  - "f is $O(g)$" means "f grows no faster than g"

$$\exists k, C > 0 \text{ such that } \forall n > k$$
$$|f(n)| \leq C|g(n)|$$

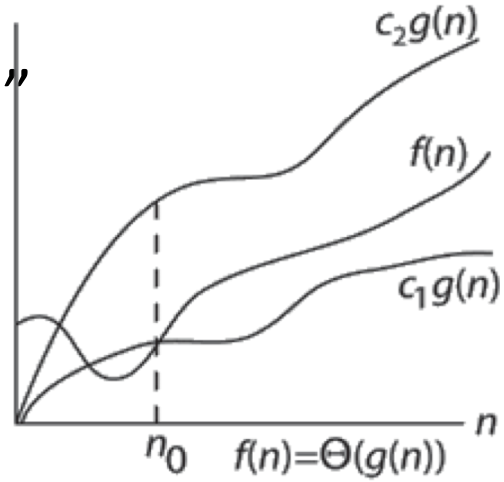


- **Big-Omega**
  - "f is $\Omega(g)$" means "f grows at least as fast as g"

$$\exists k \exists C \text{ such that } \forall n > k\text{: } |f(n)| \geq C|g(n)|$$

- ****Big-Theta****
  - "f is $\Theta(g)$" means "f grows at the same rate as g"
  - f is $\Theta(g)$ iff $f = O(g)$ and $f = \Omega(g)$

$$\exists k \exists C_1 \exists C_2 \text{ such that } \forall n > k$$
$$C_1|g(n)| \leq |f(n)| \leq C_2|g(n)|$$



Big-O vs Big-Omega vs Big-Theta

Big-O is like $\leq$:

$\mathbf{2^x = O(2^x)}$

$\mathbf{x = O(x^2)}$

$\mathbf{\log(x) = O(x^{1/2})}$

Big-Ω ("Big-Omega") is **the opposite**, like $\geq$.

$\mathbf{f(x) = O(g(x))}$
is the same thing as
$\mathbf{g(x) = \Omega(f(x))}$

$\mathbf{2^x = \Omega(2^x)}$

$\mathbf{x^2 = \Omega(x)}$

$\mathbf{x^{1/2} = \Omega(\log x)}$

Big-O vs Big-Omega vs Big-Theta

Big-O is like $\leq$:

$2^x = O(2^x)$

$x = O(x^2)$

$\log(x) = O(x^{1/2})$

Big-Θ is like $=$:

$f(x) = \Theta(g(x))$ means

$f(x) = O(g(x))$ AND

$f(x) = \Omega(g(x))$

$2^x = \Theta(2^x)$

$3x^2+2 = \Theta(x^2)$

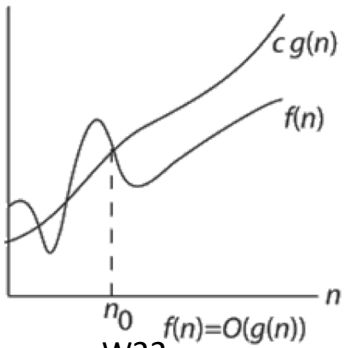
$2^{x+2} = \Theta(2^x)$

Big-Ω is like $\geq$:

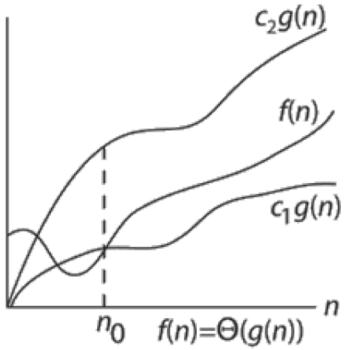
$2^x = \Omega(2^x)$

$x^2 = \Omega(x^2)$

$x^{1/2} = \Omega(\log x)$



(a)



(b)

Exercises

1. $f(n) = 5n^3$. Which are true?

(A) f = O(log n) (B) f = O(n) (C) f = O(n^2) (D) f = O((n+1)^3) (E) f = O(n^3)

2. $f(n) = n^2 + (n+1)^2 + (n+2)^2 - 100n$.

(A) f = O(log n) (B) f = O(n) (C) f = O(n^2) (D) f = O((n+1)^3) (E) f = O(n^3)

3. $f(n) = \log(5n^3)$. So f is $\Theta($ $)$.

4. $f(n) = (n+1)^3 - n^3 + 5n + 5{,}000$

So f is $\Theta($ $)$

5. Which is **not** true for the f in Question 4?

(A) f = Ω(1) (B) f = Ω(log n) (C) f = Ω(n) (D) f = Ω(n^2) (E) f = Ω(n^3)

Question 1

Let $f(n) = 5n^3$. Which of these are true?

(A) $f = O(\log n)$

(B) $f = O(n)$

(C) $f = O(n^2)$

(D) $f = O((n+1)^3)$

(E) $f = O(n^3)$

Question 1

Let $f(n) = 5n^3$. Which of these are true?

(A) $f = O(\log n)$

(B) $f = O(n)$

(C) $f = O(n^2)$

(D) $f = O((n+1)^3)$

(E) $f = O(n^3)$

Question 2

Let $f(n) = n^2 + (n+1)^2 + (n+2)^2 - 100n$. Which of these are true?

(A) $f = O(\log n)$

(B) $f = O(n)$

(C) $f = O(n^2)$

(D) $f = O((n+1)^3)$

(E) $f = O(n^3)$

Question 2

Let $f(n) = n^2 + (n+1)^2 + (n+2)^2 - 100n$. Which of these are true?

(A) $f = O(\log n)$

(B) $f = O(n)$

(C) $f = O(n^2)$

(D) $f = O((n+1)^3)$

(E) $f = O(n^3)$

This one is the **tightest** bound:
$f(n) = \Theta(n^2)$

Question 3

Let $f(n) = \log(5n^3)$ $= \log 5 + \log(n^3)$

Which of the following is true? $= \log 5 + 3\log n$

(A) $f = \Theta(\log n)$

(B) $f = \Theta(n)$

(C) $f = \Theta(n^2)$

(D) $f = \Theta((\log n)^3)$

(E) $f = \Theta(n^3)$

Question 3

Let $f(n) = \log(5n^3)$

Which of the following is true?

(A) $f = \Theta(\log n)$

(B) $f = \Theta(n)$

(C) $f = \Theta(n^2)$

(D) $f = \Theta((\log n)^3)$

(E) $f = \Theta(n^3)$

Question 4

Let $f(n) = (n+1)^3 - n^3 + 5n + 5{,}000$

Which of the following is true?

(A) $f = \Theta(\log n)$

(B) $f = \Theta(n)$

(C) $f = \Theta(n^2)$

(D) $f = \Theta((\log n)^3)$

(E) $f = \Theta(n^3)$

Question 4

Let $f(n) = (n+1)^3 - n^3 + 5n + 5{,}000$

Which of the following is true?

(A) $f = \Theta(\log n)$

(B) $f = \Theta(n)$

(C) $f = \Theta(n^2)$

(D) $f = \Theta((\log n)^3)$

(E) $f = \Theta(n^3)$

$$f(n) = 3n^2 + 8n + 5{,}001$$

Question 5

Let $f(n) = (n+1)^3 - n^3 + 5n + 5{,}000$

Which of the following is not true?

(A) f = Ω(1)

(B) f = Ω(log n)

(C) f = Ω(n)

(D) f = Ω(n^2)

(E) f = Ω(n^3)

Question 5

Let $f(n) = (n+1)^3 - n^3 + 5n + 5{,}000$

Which of the following is not true?

(A) f = Ω(1)

(B) f = Ω(log n)

(C) f = Ω(n)

(D) f = Ω(n^2)

(E) f = Ω(n^3)

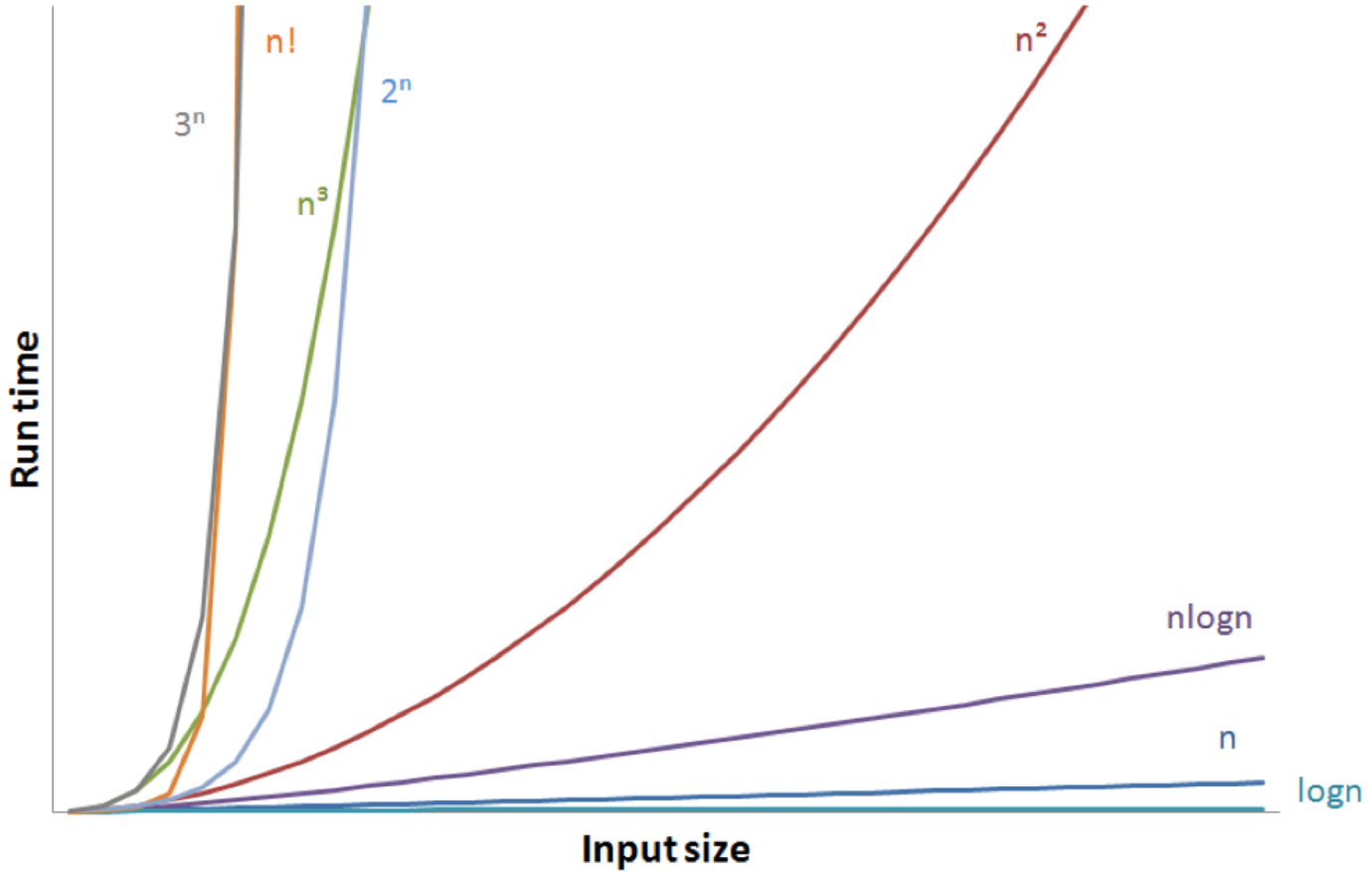
n^2

Runtime Comparison of Standard Functions

- Most functions you encounter in your career will be:
- $O(1),\ \log n,\ n,\ n\log n,\ n^2,\ n^3$ (maybe n^4,…), $2^n,\ n!$

Grows slowly (**better**)

Grows quickly (**worse**)



Runtime Comparison of Standard Functions

- Most functions you encounter in your career will be:
- $O(1),\ \log n,\ n,\ n \log n,\ n^2,\ n^3$ (maybe n^4,...), $2^n,\ n!$

Grows slowly (better)

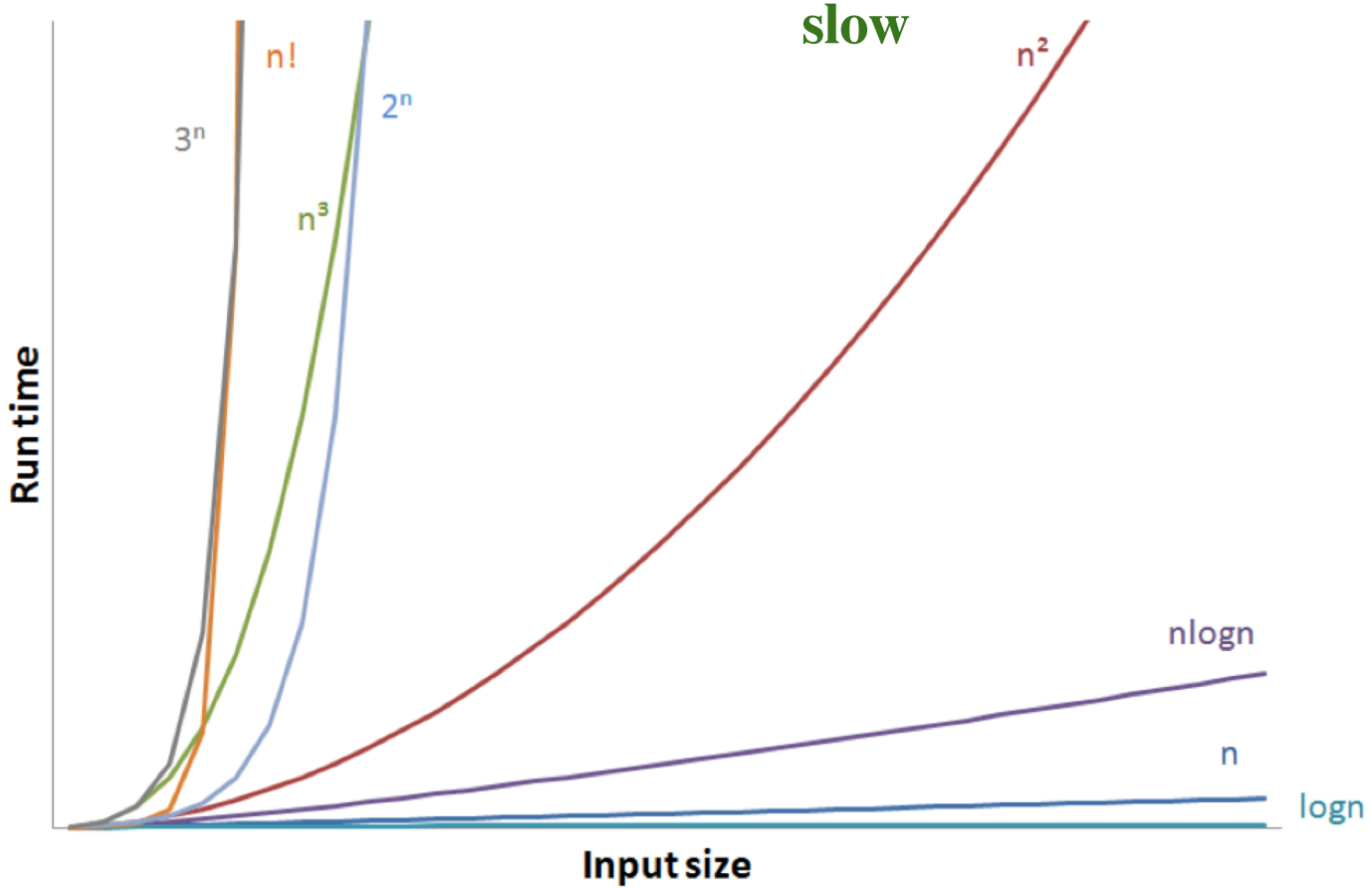
really good!

okay

usually too slow

Grows quickly (worse)

hopeless



Outline

- Recurrence and Recurrence Relations
  - Ex: Tower of Hanoi
  - Ex: Climbing stairs
  - Ex: Katie's Pandemic Outfits
- Big-O Notation
  - Introduction & Definitions (Big-O, Big-Omega, Big-Theta)
  - Examples: Runtime of mathematical functions
  - **Properties for Combining Functions**
- Next time: Big-O and recurrence relations come together

Properties for Adding and Multiplying

Consider positive-valued functions $f_1(n) = \Theta(g_1(n))$ and $f_2(n) = \Theta(g_2(n))$.

- **Addition**

$$(f_1 + f_2)(n) = \Theta(\max(g_1(n), g_2(n)))$$

- **Scalar multiplication**

$$af(n) = \Theta(f(n))$$

- **Product**

$$(f_1 \cdot f_2)(n) = \Theta(g_1(n) \cdot g_2(n))$$

Big-Theta "Cheat Sheet"

Runtime comparison of standard functions

$\Theta(1),\ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ (\text{maybe } n^4, \ldots),\ 2^n,\ n!$

Grows slowly (_____)

Grows quickly (_____)

Consider positive-valued functions $f_1(n) = \Theta(g_1(n))$ and $f_2(n) = \Theta(g_2(n))$.

- **Addition**

$$(f_1 + f_2)(n) = \Theta(\quad)$$

- **Scalar multiplication**

$$af(n) =$$

- **Product**

$$(f_1 \cdot f_2)(n) = \Theta(\quad)$$

Examples: Runtimes Galore!

Determine the Θ estimate for each of the following functions.

a) $f(n) = (n^3 + n^2)(n^2 + 50{,}000)$

b) $f(n) = \left(5n + \frac{n}{2} + 7\right)(3n^4 + 8n!)$

c) $f(n) = (n^5 + 2^n + \log n)(3n + 4n \log n)$

d) $f(n) = 10n(\log n^2 + \log n^3)(n^2 + 1)$

e) $f(n) = (3^n + n!)(2^n + n^2)$

f) $f(n) = (3^n + n!) + (2^n + n^2)$

g) $f(n) = n^2 \cdot 3^n + \frac{n}{3} \cdot n^3 \log n$

Examples: Runtimes Galore!

Determine the Θ estimate for each of the following functions.

a) $f(n) = (n^3 + n^2)(n^2 + 50{,}000)$

n^3 term dominates; n^2 term dominates

Using the product rule for complexity, f is $\Theta(n^3 \cdot n^2)$, i.e., f is $\Theta(n^5)$

b) $f(n) = \left(5n + \frac{n}{2} + 7\right)(3n^4 + 8n!)$

n; $n!$

f is $\Theta(n!\, n)$

c) $f(n) = (n^5 + 2^n + \log n)(3n + 4n \log n)$

2^n; $n \log n$

f is $\Theta(n 2^n \log n)$

d) $f(n) = 10n(\log n^2 + \log n^3)(n^2 + 1)$

n; n^2; $(2 \log n + 3 \log n)$; $\log n$

f is $\Theta(n \cdot \log n \cdot n^2)$, so f is $\Theta(n^3 \log n)$

e) $f(n) = (3^n + n!)(2^n + n^2)$

$n!$; 2^n

f is $\Theta(n!\, 2^n)$

f) $f(n) = (3^n + n!) + (2^n + n^2)$

$n!$; 2^n

f is $\Theta(\max(n!,\ 2^n))$ so f is $\Theta(n!)$

g) $f(n) = n^2 \cdot 3^n + \frac{n}{3} \cdot n^3 \log n$

$\frac{1}{4} n^4 \log n$

f is $\Theta(\max(n^2 3^n,\ n^4 \log n))$, so f is $\Theta(n^2 3^n)$

More Examples

Consider positive-valued functions f and g, where

f is $\Theta(n^2)$ and g is $\Theta(n \log n)$.

Determine the Θ estimate for each of the following functions.

a) $h(n) = 2f(n) + g(n)$

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

b) $h(n) = 36f^4(n)$

d) $h(n) = f^2(n) + 10{,}000f(n)g^3(n)$

More Examples

Consider positive-valued functions f and g, where

f is $\Theta(n^2)$ and g is $\Theta(n \log n)$.

Determine the Θ estimate for each of the following functions.

a) $h(n) = 2f(n) + g(n)$

- $2f$ is $\Theta(n^2)$
- h is $\Theta((\max(n^2, n \log n))$
- so h is $\Theta(n^2)$

b) $h(n) = 36f^4(n)$

- $36f^4$ is $\Theta(f^4)$
- So h is $\Theta(f^4)$
- which is $\Theta((n^2)^4)$
- which is $\Theta(n^8)$

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

- f^2 is $\Theta((n^2)^2)$, i.e., $\Theta(n^4)$
- h is $\Theta(n^4 \cdot n \log n)$
- so h is $\Theta(n^5 \log n)$

d) $h(n) = f^2(n) + 10{,}000f(n)g(n)$

- f^2 is $\Theta((n^2)^2)$, i.e., $\Theta(n^4)$
- $1000f(n)g(n)$ is $\Theta(n^2 \cdot n \log n)$
- which is $\Theta(n^3 \log n)$
- h is $\Theta((\max(n^4,\ n^3 \log n))$
- which is $\Theta(n^4)$

Even More Examples

Consider functions f and g with $f, g \geq 0\ \forall n$ where:

- f is $O(n^3)$ and $\Omega(n)$
- g is $\Theta(\log n)$

Fill in the blanks by finding the largest lower bound and smallest upper bound on h_1 and h_2.

a) If $\boldsymbol{f + g}$ is $\Theta(h_1(n))$, then _____ $\leq h_1(n) \leq$ _____

b) If $\boldsymbol{f \cdot g}$ is $\Theta(h_2(n))$, then _____ $\leq h_2(n) \leq$ _____

Even More Examples

Consider functions f and g with $f, g \geq 0\ \forall n$ where:

- f is $O(n^3)$ and $\Omega(n)$
- g is $\Theta(\log n)$

Fill in the blanks by finding the largest lower bound and smallest upper bound on h_1 and h_2.

a) If $\boldsymbol{f + g}$ is $\Theta(h_1(n))$, then $\underline{\quad n \quad} \leq h_1(n) \leq \underline{\quad n^3 \quad}$

b) If $\boldsymbol{f \cdot g}$ is $\Theta(h_2(n))$, then $\underline{\quad n \log n \quad} \leq h_2(n) \leq \underline{\quad n^3 \log n \quad}$

Wrap-up

- Recursion is everywhere (physical processes, mathematical modeling, programs)
  - Today we saw a number of *linear* recurrences
- "Big-Theta" is a measure of the runtime of an algorithm
  - We only care about the behavior as the input gets huge
  - "Big-O" gives an upper bound on runtime
  - "Big-Omega" gives a lower bound on runtime
- Next time
  - Runtimes of algorithms based on their pseudocode
  - Divide-and-Conquer recurrences, and their runtimes