

Transition: DS + Algorithms

Big-O
+ Recursion



First:

worksheets from last time?

1) Basis: $0 \in S + 1 \in S$
Recursive: if $w \in S$ then $\begin{matrix} 0w0 \\ 0w1 \\ 1w0 \\ 1w1 \end{matrix} \left. \vphantom{\begin{matrix} 0w0 \\ 0w1 \\ 1w0 \\ 1w1 \end{matrix}} \right\} \in S$ also

S is an infinite set!

$S = \{ \underbrace{0}_{w=0}, \underbrace{1}_{w=0}, \underbrace{000}_{w=0}, \underbrace{001}_{w=0}, \underbrace{100}_{w=0}, \underbrace{101}_{w=0} \}$

let

$w=0$

let $w=1$: add 010, 011, 110, 111,

let $w=00$, 00000, 00001, 10000, 10001,

let $w=001$, _____

let $w=00000$

Prove any $w \in S$ has odd length.

inducting on size of $w = |w|$
Base case: only length 1 strings

in S are $0 + 1$

$\hookrightarrow$ are no length 2 strings, because
any string added by rec part
has length ≥ 3 .

PH: for any $w' \in S$ with $|w'| < k$,
then $|w'|$ is odd.

IS: Consider $w \in S$ with $|w| = k > 1$
 w had to be created using recursive
construction so w is either:
 $\begin{matrix} 0w'0 & 1w'0 \\ 0w'1 & 1w'1 \end{matrix}$ for a $w' \in S$.

Since $w' \in S$ and $|w'| < |w|$,
IH says $|w'|$ is odd.

Then $|w'| = 2l+1$, $l \in \mathbb{Z}$

$$\begin{aligned}\text{Now: } |w| &= 2l+1 + 2 \\ &= \underbrace{2(l+1)}_{\in \mathbb{Z}} + 1\end{aligned}$$

$|w|$ is odd length



Alternative:
 induct on $n = \#$ times recursive rule
 is used to build $w \in S$.

BC: 0 times $\rightarrow 0, 1 \in S$
 both are odd

IH: $\leq k$ times, then ~~string~~ w' is odd length

w' is gen. using rec rule
 IS: $k+1$ times:

$w =$
 $0w'0$
 $0w'1$
 $1w'0$
 $1w'1$

where w' had
 fewer rec rules
 applied

by IH, w' has odd length
 $\rightarrow |w| = |w'| + 2$ so also odd.

if x & y are strings,
 xy is x concatenated with y

"hello" + "!" = "hello!"

$$x + y = x$$

"hello" + " = "hello"



ϵ , the empty string

"hello" + ϵ + "!" = "hello!"

"_ " = $\neq \epsilon$

Build set P: and $\{ \epsilon \}$

basis:

$\epsilon \in P$

rec: if $x, y \in P$, then $(x)y \in P$
 $"(" + x + ")" + y$

(infinite)

$$P = \{ \epsilon, \underline{()}, ()(), (()), (())(), \dots \}$$

set $x=y=\epsilon$, $"(a)\epsilon" = \underbrace{()}_\epsilon =$

② $\begin{matrix} x=\epsilon \\ y=() \end{matrix} : \begin{matrix} \overset{x}{()} \overset{y}{()} \\ \hline \end{matrix} = ()()$

$\begin{matrix} x=() \\ y=\epsilon \end{matrix} : \begin{matrix} \overset{x}{()} \cdot \epsilon \\ \hline \end{matrix}$

$\begin{matrix} x=() \\ y=() \end{matrix} : \begin{matrix} \overset{x}{()} \overset{y}{()} \\ \hline \end{matrix}$

Induction:

$$\left(\begin{array}{l} P(0) \\ \forall n \geq 0, P(n) \rightarrow P(n+1) \end{array} \right)$$

Stronger:

$$P(0)$$

$$\text{if } \left(\forall k \leq n \right)$$

$$P(k) \rightarrow P(n+1)$$

Show: every string in P has $\#left = \#right$

Induct on length of string:

Base case: $\epsilon \in P$, so $|w| = 0$ (length 0)
 0 left $\checkmark$ 0 right $\checkmark$

IH: Assume any string $w' \in P$ with
 $|w'| \leq k$ has same $\#$ of $(\&)$.

IS: Consider any $w \in P$ of length $k > 0$.
Know w came from rec construction:

some $x, y \in P$ s.t. $w = (x)y$

Now, $|w| = |(x)y| = k$ so $|x| \neq |y|$

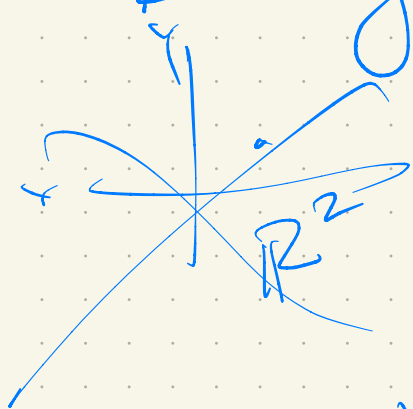
are both $< k$

$\Rightarrow$ by IH, both x & y have same
 $\#$ of $(\&)$.

say x has a (and a)
 y has b (and b)
some $a, b \in \mathbb{Z}^+$,

then $w = \underline{(x)} y$
so w has $\underline{1} + a + b$ of $($
and $\underline{1} + a + b$ of $)$
 $\Rightarrow$ same # of (and) $\square$

Binary trees:



• node
or vertex

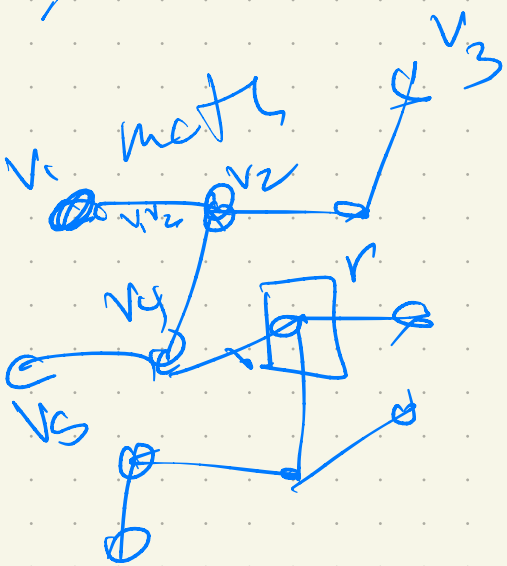
Graph theory:

$$G = (V, E)$$

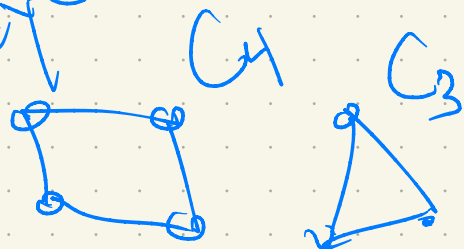
↑ vertices ↑ edges

V objects: $\{v_1, \dots, v_n\}$

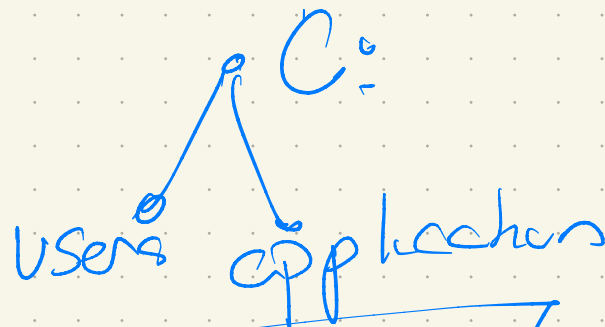
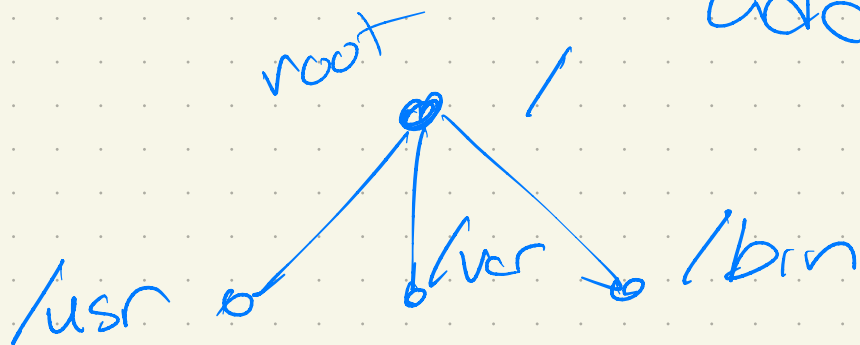
E is pairs $\{v_i, v_j\}$,
 $v_i, v_j \in V$



graph
with
no cycles

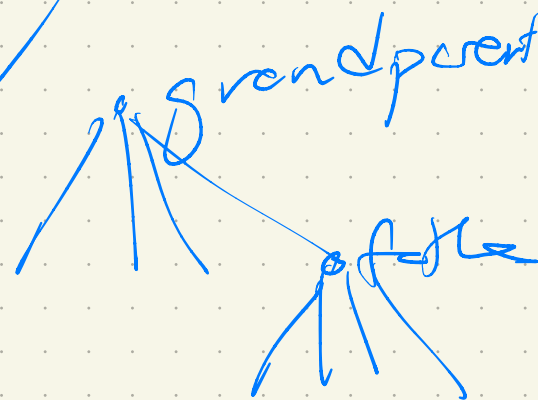
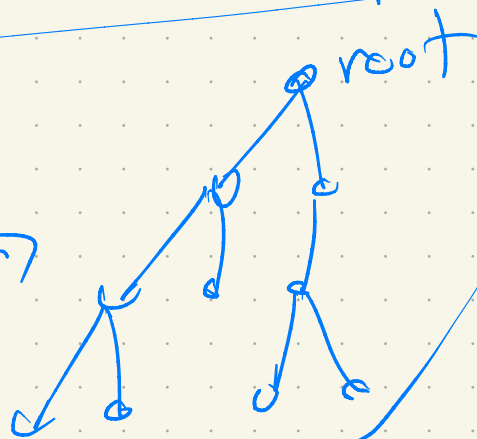


Rooted tree: not adj $\{v_i, v_j\}$
add parent / child



Rooted binary tree:

every node has ≤ 2 children

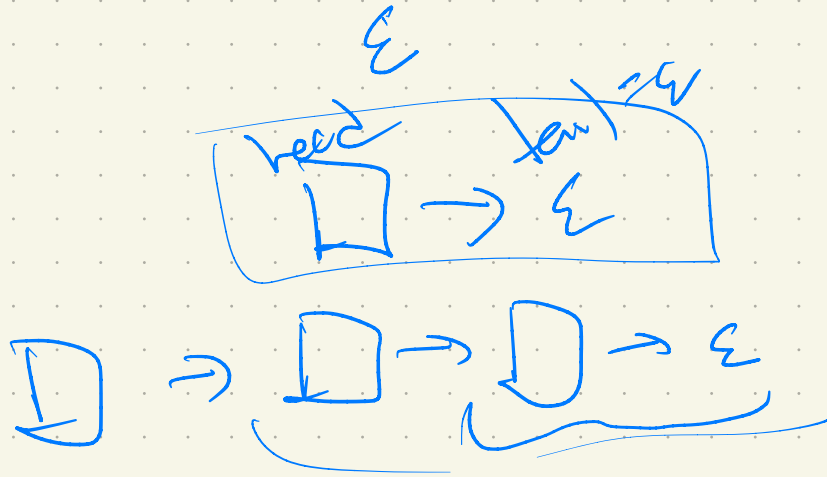


Graph with n root
 ≤ 2 children per node

lists:

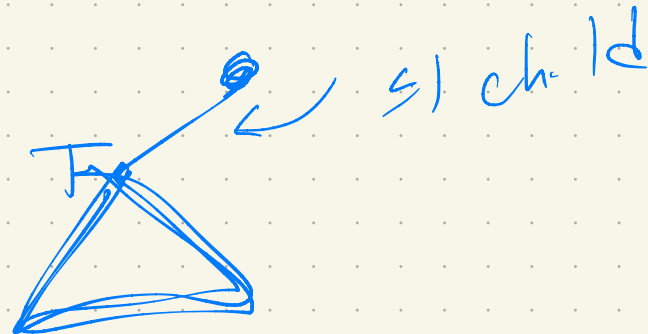
rec: basis: empty list

rec: one element (head)
+ a list after it (tail)

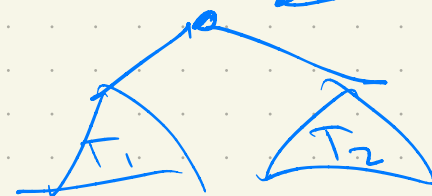


binary tree: B
 basis: root

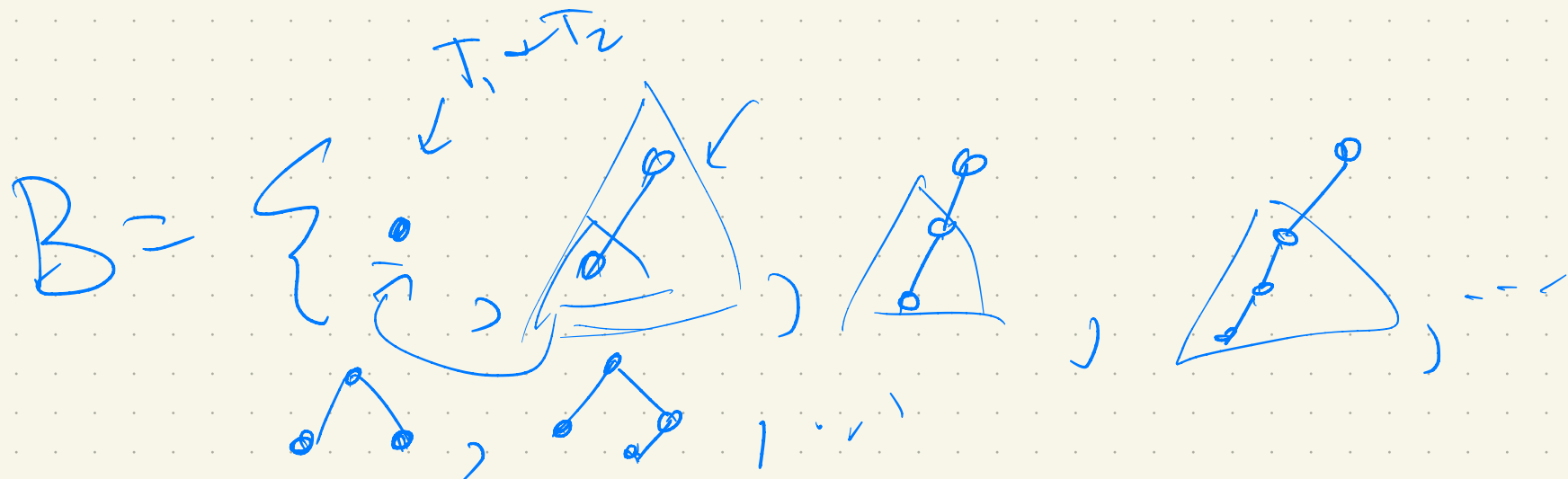
rec 1:



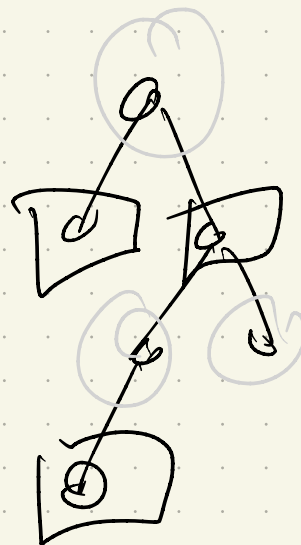
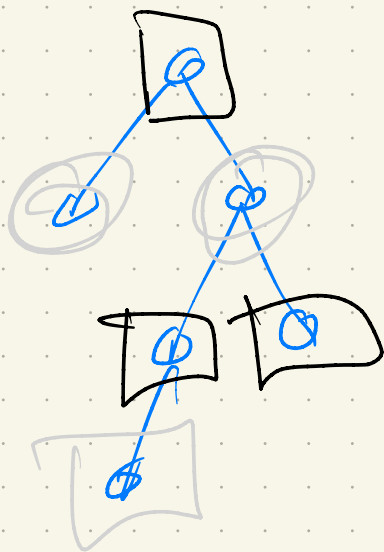
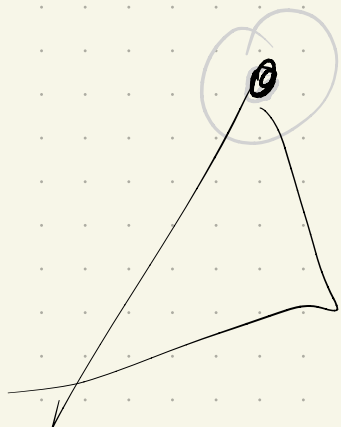
rec 2:



apply 1 & 2 repeatedly $\rightarrow$ infinite set



0



Next: Confusing readings!

Let's pause on recursion -

I'll post something else covering it soon.

Back to Big-O & why we care!

Fib: if $n=0$ or 1 , $T(n) = \boxed{O(1)} \approx \underline{5} = C$

if $n \geq 2$: $T(n) =$ runtime of your fib fn

$$T(n) = 3 + T(n-1) + T(n-2)$$

$$T(n) = T(n-1) + T(n-2) + \underline{3}$$

$$T(n) = T(n-1) + \underbrace{T(n-2)}_{\substack{\swarrow \downarrow \\ T(n-2)+T(n-3)+3}} + 3$$

$$T(n-2) + T(n-3) + 3 \quad T(n-3) + T(n-4) + 3$$

$$= T(n-2) + 2T(n-3) + T(n-4) + 9$$

$$\underbrace{T(n-2)}_{T(n-3)+T(n-4)+3} \quad \underbrace{2T(n-3)}_{2(T(n-4)+1)}$$

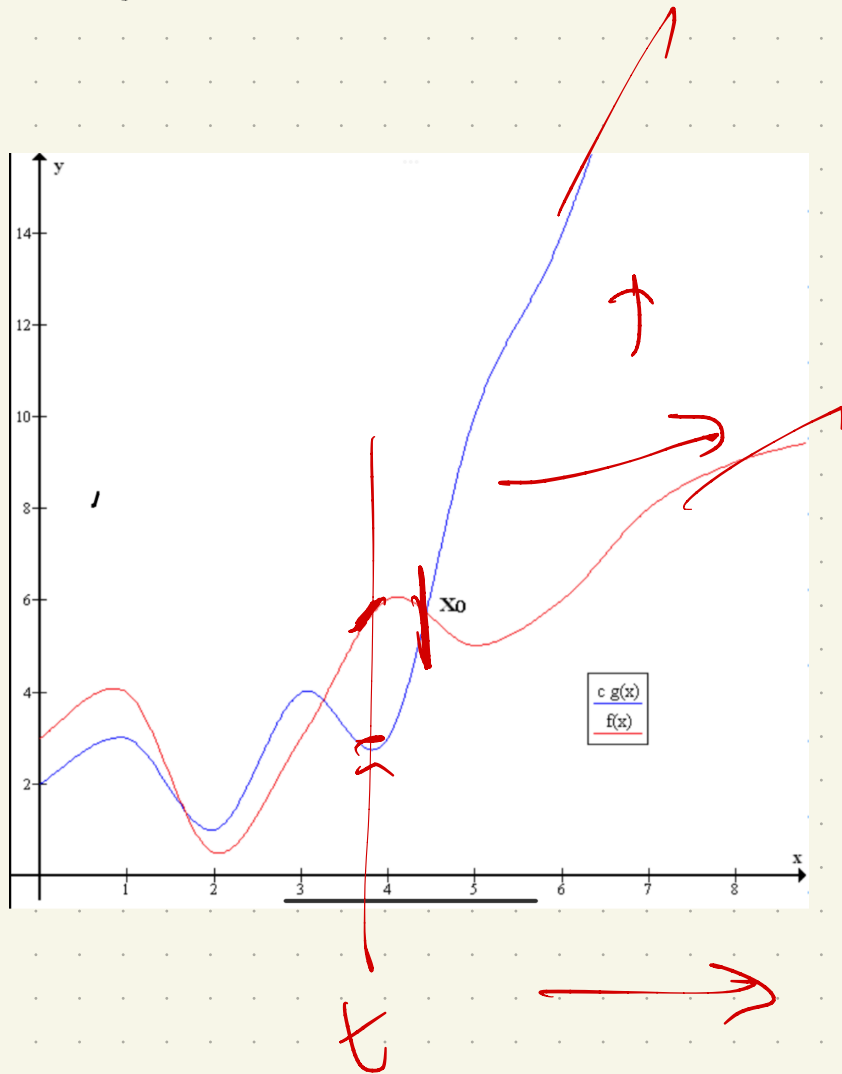
...

$$\Rightarrow 3 + 3 + 3 + \dots + 3$$

??

Asymptotic Analysis

How can we compare functions?



1) Fix a value t , & compare $f(t)$ to $g(t)$

"

big-O view:
zoom out (t larger)

Is there a point
where one really is
bigger

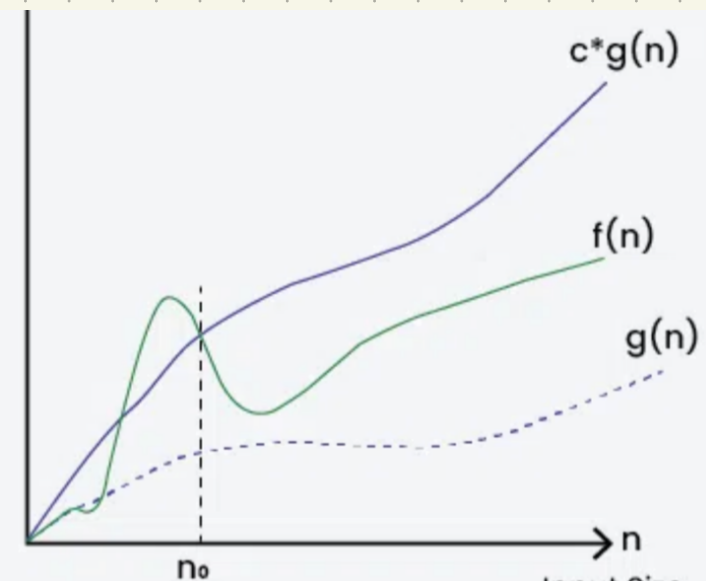
Big-O

Let f and g be functions from

$\mathbb{Z}^+$ to $\mathbb{R}^+$
why?

We say $f(n) = O(g(n))$ if $\exists$
constants c and n_0 s.t. $\forall n > n_0$,
 $f(n) \leq c \cdot g(n)$

Note: $O(g(n))$ is
a set.



Example

$$f(x) = x^2 + 2x + 1 \quad \text{is} \quad O(x^2)$$

proof: Find c and n_0 , then show inequality holds.

General strategy

- Start with n_0 : pick it so can estimate size of $f(n)$ for $n > n_0$.
 - ↳ often based on coefficients

- Then look for a constant c so inequality holds.

Example: Show $f(x) = 5x^2 + 25x + 3$ is $O(x^3)$.

We'll write $f(x) = O(g(x))$.

NOT AN EQUALITY!

• $x^2 + 2x + 1$ is $O(x^2)$

• $x^2 + 2x + 1$ is $O(x^3)$

Really, means $f(x) \in \{ \text{functions that are } O(g(x)) \}$

Polynomials are usually nice $\rightarrow$ but
Can do many functions!

Ex: $\sin x$ is $O(1)$.

Ex: n is $O(n \log_2 n)$.

Ex Show that n^2 is not $O(n)$

proof: harder! need to show

$$\neg (\exists c \exists n_0 \text{ s.t. } \forall n > n_0, n^2 \leq c \cdot n)$$



$\forall c$ and $\forall n_0$, $\exists$ some $n > n_0$ with

So:

Why do we care??

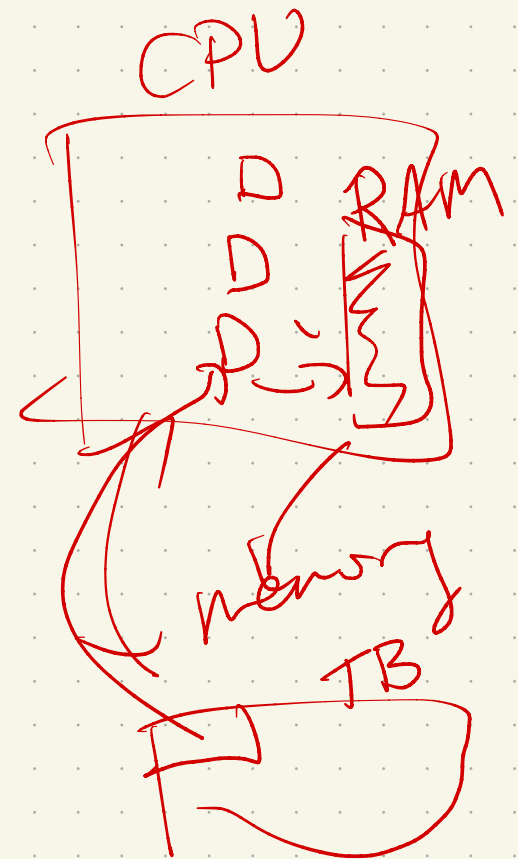
Back to pseudocode (or code)

We usually formalize the running time of an algorithm in terms of the number of "atomic operations."

Examples:

↳ "1 clock cycle"

- load values to register
- add 2 values
- compare + "branch"
- subtracting
- multiplication
- ...



Why count operations?

Generally, the other way to compare speed is benchmarking.

But this can vary even with identical algorithms:

- different languages
- different hardware
- different compilers
- different network conditions
- different CPU load
- different data ..

} Context

Ideally: we'd like an abstract way to compare only the algorithms, regardless of context.

Why big-O?

The same "operation" in a language (ie a loop or function call) can use different low level operations, depending upon language, compiler, processor, etc.

Also, look at rates:

Table 7.3.1: Growth rates for different input sizes.

$f(n)$	$n = 10$	$n = 50$	$n = 100$	$n = 1000$	$n = 10000$	$n = 100000$
$\log_2 n$	$3.3 \mu s$	$5.6 \mu s$	$6.6 \mu s$	$10.0 \mu s$	$13.3 \mu s$	$16.6 \mu s$
n	$10 \mu s$	$50 \mu s$	$100 \mu s$	$1000 \mu s$	10 ms	0.1 s
$n \log_2 n$	0.03 ms	0.28 ms	0.66 ms	10.0 ms	0.133 s	1.67 s
n^2	0.1 ms	2.5 ms	10 ms	1 s	100 s	2.8 hours
n^3	1 ms	0.125 s	1 s	16.7 min	11.6 days	31.7 years
2^n	1.0 ms	35.7 years	$4.0 \times 10^{16} \text{ years}$	$3.4 \times 10^{287} \text{ years}$	$6.3 \times 10^{2996} \text{ years}$	*

This data assumes that a single instruction takes $1 \mu s$ to execute. A μs is a microsecond and is equal to 10^{-6} seconds. An ms is a millisecond and is equal to 10^{-3} seconds.

Consider:

$5n \log n$ versus n^2 , like when
sorting a list of 10,000 people

That limit definition

I did $f(n) = O(g(n))$ if $\exists n_0 \ \& \ c$ s.t.

$$\forall n > n_0, \quad f(n) \leq c \cdot g(n)$$

Take that inequality:

$$\text{So } f(n) = O(g(n)) \Leftrightarrow$$

Big Ω

Let f and g be functions from $\mathbb{Z}^+ \rightarrow \mathbb{R}^{\geq}$

We say f is $\Omega(g(n))$ if

$\exists$ positive constants c and n_0
such that $\forall n > n_0$,

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

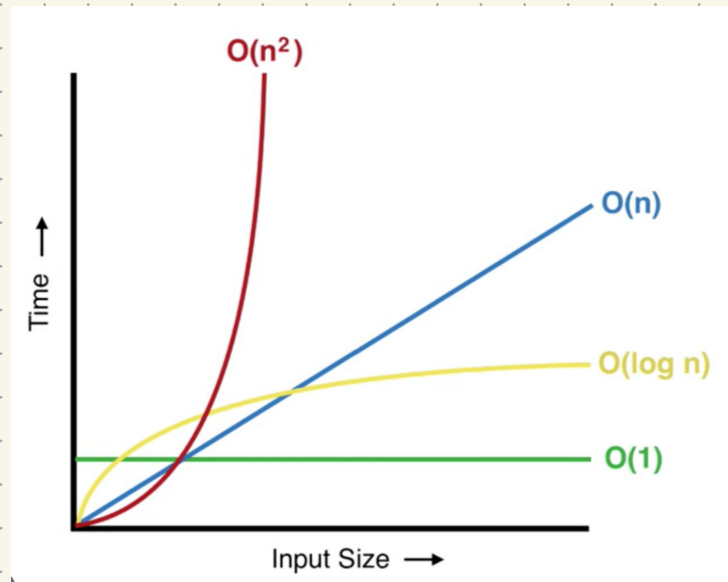
" f is big-omega of g "

Example: $5 \cdot n$ is $\Omega(n)$:

Big- Θ

If f is $O(g)$ and $\Omega(g)$ then f is $\Theta(g)$.

This gives a way to nicely "bucket" function size:



Theorems

Let $p(n)$ be a degree k polynomial, so

$$p(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$$

for constants a_i and $a_k \neq 0$.

Then $p(n) = \Theta(n^k)$.

Proof:

Big- Θ :

Big-O:

Using this theorem;

Give big- Θ estimates for:

- $f(x) = \frac{1}{8} x^5 + x^3 + 2$

- $f(x) = 20000 x^2 - 100000 x$

- $f(x) = \frac{2x^3}{3000} - x + 2$