An algorithm is a finite group of instructions that provides a step-by-step procedure for solving a problem involving...

- Computation
    - Arithmetic results
    - Matrix operations
    - Numerical analysis - improving accuracy
- Data Organization
    - Searching
    - Sorting (alphabetization, ordering a group of #s)
- Optimization
    - Using mathematical models to find the best solution to a real-world problem
    - Graph theory
        - find shortest paths, efficient networks.
    - Knapsack Problem

What is an "efficient" algorithm?

- Fast
- Takes up little memory/space

Let n = the size of the data set of interest
not formal
input for the algorithm

Ex Searching, Sorting: n = # items in a database
Ex n could be the "size" of a square matrix in a computation problem

Let $f(n)$ = the "amount" of time or memory used by a particular algorithm to deal with the "worst-case scenario" involving a data set of size n.

Pessimistic

"Worst-case" analyses tend to be easier; more commonly used than other analyses.
Also: "Best-case", "Average-case"
more useful, but harder
Need to assign probs. for the possible inputs. Maybe hard even for uniform

A single, really bad "pathological case" might skew a worst-case analysis "unfairly."

n	$f(n)$		
1	12	← worst-case for data size	1
2	40	←	2
3	84	←	3

Let's say $f(n)$ = worst-case running time for size-n data

Assume $f(n), g(n), \ldots$ are always > 0.

How do we measure running times?
    Actual time in seconds, hours, ...
        on a specific computer.
    # of simple program statements/condition checks/
        comparisons (searching/sorting), etc.

We want $f(n)$ to be low.

Examples

	In practice	In theory
① $f(n) = 2$ (seconds) for all $n \in \mathbb{Z}^+$ The worst-case running time is a nice constant function of n. (≤ 2 for all data sets) We wish!	Great!	Great!
② $f(n) = 2 + \underbrace{\lfloor \sqrt[10]{n} \rfloor}_{\text{grows very slooowly with } n}$	Great!	Great!

1 mil secs ≈
12 days?

③ $$f(n) = \underbrace{1\text{ million}}_{\substack{\text{expensive}\\ \text{fixed/start-up}\\ \text{cost}}} + \lfloor \sqrt[10]{n} \rfloor$$

Yuk! Great!

Focuses on
the slow <u>growth</u>

④ $$f(n) = 1\text{ million} + \underbrace{(1\text{ billion})}\lfloor \sqrt[10]{n} \rfloor$$

Yuk! Great!

This factor is
less relevant as
$n \to \infty$. "Not much"
when $n = 10^{10,000}$
relatively speaking!

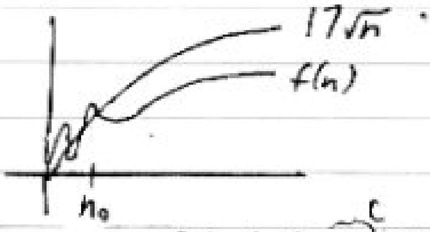
"Big-O" Notation

$f(n)$ is $O(g(n)) \Leftrightarrow$

f is *eventually* bounded above by some constant multiple of $g(n)$ $\Leftrightarrow$

$f(n) \le C g(n)$ for some constant C and for "large enough" n (i.e., $n > n_0$ for some n_0)

Ex



$f(n) \le \overbrace{17}^{C}\sqrt{n}$ for all $n > n_0$

So, $f(n)$ is $O(\sqrt{n})$

Big-O: "upper bound idea"
Big-Ω (omega): "lower bound idea" $17\sqrt{n}$ is $\Omega(f(n))$
Big-Θ (theta): combines the two

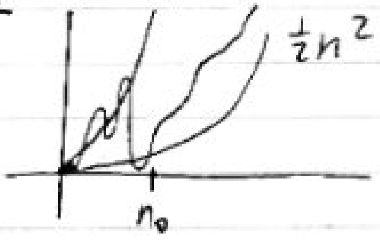
$f(n)$ is $\theta(g(n)) \iff$

f is eventually bounded above and below by constant multiples of $g(n) \iff$

$$\underbrace{C_1 g(n) \le f(n)}_{f(n) \text{ is } \Omega(g(n))} \underbrace{\le C_2 g(n)}_{f(n) \text{ is } O(g(n))}$$

for some "constants" C_1, C_2
and for "large enough" n

Ex



$f(n)$ is $\theta(n^2)$
We say that $f(n)$ is "order" n^2.

Since we often use worst-case analyses,
"O" is especially useful.

Well-known complexity classes:
(from best to worst)

$O(1)$	constant complexity (like $f(n)=2$)
$O(\log n)$	logarithmic $O(\log_2 n) = O(\log_{10} n) = O(\ln n) = O(\log_b n),\ b>1$
$O(n^b),\ 0<b<1$	Ex $n^{1/2}=\sqrt{n},\ n^{1/3}=\sqrt[3]{n}, \dots$
$O(n)$	linear
$O(n \log n)$	
$O(n^b),\ b>1$	polynomial Ex $\underbrace{4n^3}_{\text{focus on leading term}}+6n^2-n+2$ is $O(n^3)$, (it's also $O(n^4)$, $O(2^n)$, etc. We want the "tightest" func. we can find.
$O(b^n),\ b>1$	exponential
$O(n!)$	
$O(n^n)$	

$O(1) \subset O(\log n) \subset O(n^6) : O(6^n) \subset \ldots$

$\longrightarrow$

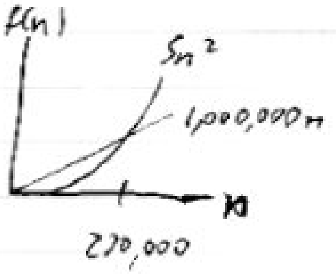
Theory: worse
(as $n \to \infty$: asymptotic behavior)

Which is better:

$f(n) = 5n^2$ is $O(n^2)$
Better for small n

$g(n) = 1{,}000{,}000\,n$ is $O(n)$
Eventually ($n > 200{,}000$), this is better.

$5n^2 = (1M)n$
$= 2 \times 10^{11}$
at $n = 100{,}000$



(scales different)

Rules for O

Sum Ex $\underbrace{3n^4}_{O(n^4)} + \underbrace{7n^2}_{O(n^2)} + \underbrace{5\log n}_{O(\log n)}$ is $O(n^4)$

↑

Take the "biggest" function in the sum

Product Ex

$$\underbrace{(2^n + n^5)}_{O(2^n)}\underbrace{(3n-1)}_{O(n)} \quad \text{is} \quad O(n2^n)$$

When we multiply "pieces",
we multiply their O-functions.

Note "Θ" means "order"

Ex $\underbrace{4n^2+2n}$ is $\Theta(n^2)$

are "roughly comparable"

Sloppy books often say "O" means "order."

Cute Proofs

Ex 4 $f(n) = 1+2+\ldots+n$
$\leq n+n+\ldots+n$
$= n^2$

$f(n) \leq n^2$ In fact, true for all $n \in \mathbb{Z}^+$

So, $f(n)$ is $O(n^2)$

In fact, $1+2+\ldots+n = \cancel{\frac{n+1}{(2}} \cdot \frac{1+n}{2}(n)$

$$= \frac{n(n+1)}{2}$$

$$= \frac{1}{2}n^2 + \frac{1}{2}n$$

So, $f(n)$ is $\Theta(n^2)$, also.

Ex 5 $f(n) = n!$

$n! = 1 \cdot 2 \cdot 3 \cdots n$
$\leq n \cdot n \cdot n \cdots n$
$= n^n$

So, $n! \leq n^n$ In fact, true for all $n \in \mathbb{Z}^+$

So, $\overset{n!}{f(n)}$ is $O(n^n)$

In fact, (Stirling's approx.)

$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \underbrace{(1+O(\tfrac{1}{n}))}_{\text{can omit}}$ for big n

useful for proofs/computs w/ $n!$

SEARCHING ALGORITHMS

Linear

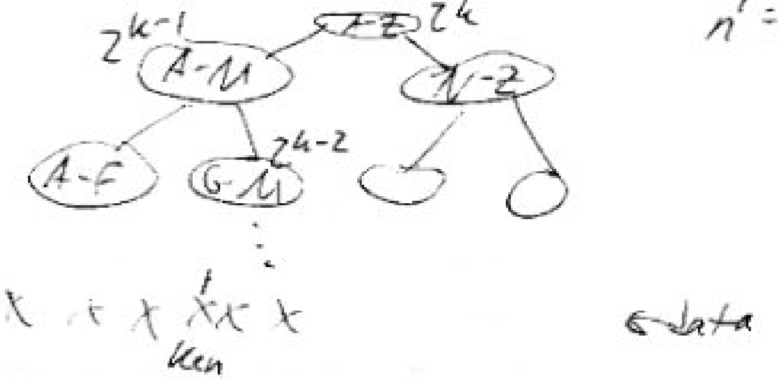
Look for x

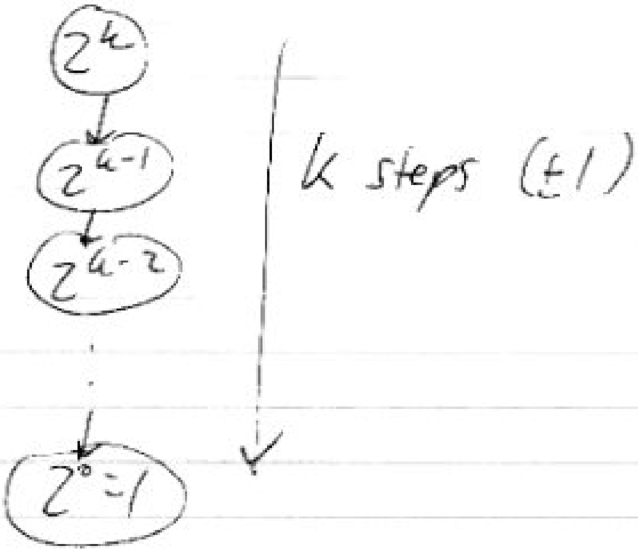
$O(n)$

≈ same amt. of work at each step

Binary search (assume sorted already!)

$n = 2^k$





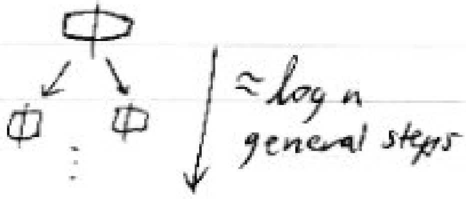
$n = 2^k$
$\Rightarrow k = \log n$

$O(\log n)$

9.3 Math 101

Searching a sorted array – $\Theta(\log n)$

Ex Binary Search



Searching an unsorted array – $\Theta(n)$

Ex Linear Search

Sorting an array – $\Theta(n \log n)$

Ex Mergesort