CS25-X94 L1 P4

Strassen's Algorithm

Multiply $n \times n$ matrices
$C = AB$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \text{ for } i,j = 1, \ldots, n$$

Ordinary method:
n^2 entries, n multiplications/entry
$\Rightarrow n^3$ mults
$\Rightarrow \Theta(n^3)$ time (explain Θ)

Strassen's idea:
Multiply 2×2 matrices with only 7 multiplications instead of 8.

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

add	mult	
1	1	$P_1 = a \cdot (g - h)$
1	1	$P_2 = (a + b) \cdot h$
1	1	$P_3 = (c + d) \cdot e$
1	1	$P_4 = d \cdot (f - e)$
2	1	$P_5 = (a + d) \cdot (e + h)$
2	1	$P_6 = (b - d) \cdot (f + h)$
2	1	$P_7 = (a - c) \cdot (e + g)$
10	7	

add	
3	$r = P_5 + P_4 - P_2 + P_6$
1	$s = P_1 + P_2$
1	$t = P_3 + P_4$
3	$u = P_5 + P_1 - P_3 - P_7$

CS25 - X94 L1 P5

Demo: $s = P_1 + P_2$
$= ag - ah + ah + bh$
$= ag + bh$
(Work out other 3 on your own.)

Count additions (18), mults (7).

Doesn't rely on commutativity of mult
$\Rightarrow$ can use these same equations for submatrices.

In fact, it only pays to use them for submatrices. For scalars, 14 new adds to eliminate 1 mult is a bad trade off.

Recursive alg:

D&C paradigm
1. Divide: Partition A, B into $\frac{n}{2} \times \frac{n}{2}$ matrices.
2. Conquer: Perform 7 multiplications recursively.
3. Combine: Form C using above eqns.

Get recurrence for running time.
Let $T(n)$ = time to multiply 2 $n \times n$ matrices

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 7T\left(\frac{n}{2}\right) + \Theta(n^2) & \text{if } n > 1. \end{cases}$$

Solution (next 2 classes):
$T(n) = \Theta(n^{\lg 7}) = O(n^{2.81})$.

CS 25 - Algorithms

9/22/95

Last time

- Administrivia
- Strassen's alg / D&C paradigm

Today

- Analysis of alg.
- Order notation
- Recurrences

Handouts

- Course Info II
- Homework 1

- **Analysis of Algorithms** (context for the moment - sorting)

Actual running time depends on:

will focus on these {
- input size, e.g. 10 vs 10,000
- input itself, e.g. partially sorted already, etc.

- implementation, good vs. poor
- machine itself, fast vs. slow

Kinds of Analysis:

(usual) - worst case: $T(n) =$ max time on any input of size n

(sometimes) - average case: $T(n) =$ average time over all inputs of size n (assumes dist. over inputs)

(never) - best case: $T(n) =$ min time over any input of size n

Will focus on worst case generally (explain why), average case sometimes.

$\Downarrow$

worst case analysis
- more general than average case analysis
- usually much simpler than a.c.a.
- often obtain competitive results as compared to a.c.a
- " " known lower bounds

(over)

How to frame worst-case results?

Asymptotics

- ignore small input sizes; focus on arbitrarily large n
  - can write special-purpose algs for constant-sized inputs (e.g. quicksort example w/ insertion sort for small inputs)
- ignore constant factors (mostly)
  - can combat constant factors by - better implementation
    - faster machines.
  - can't combat bad asymptotics...

How do we denote asymptotic results?

Asymptotic (or order) notation

~~CS25-X94 Lecture 2 7/1/94~~

~~No class Monday – July 4 holiday.~~
~~Using x-hours for the next 2 weeks –~~
~~Tuesday 1:00–1:50.~~

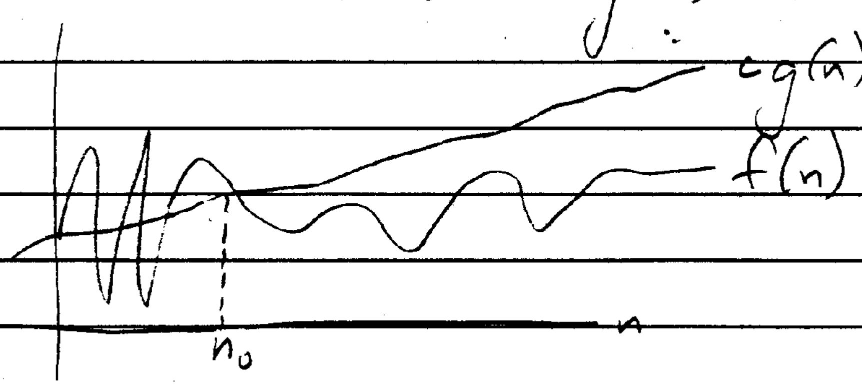
Asymptotic notation

How we characterize the order of growth of functions.

~~Could determine exact running time for an alg, but it depends heavily on the impl.~~
~~More interesting to study how running time increases with input size.~~

(upper bounds) O-notation $\Rightarrow$ $f(n) = O(g(n))$ if $\exists$ const. $c, n_0 > 0 \ni$
$0 \le f(n) \le c\,g(n) \;\; \forall n \ge n_0$. - Funny;

$$O(g(n)) = \{f(n) : \exists \text{ consts } c, n_0 > 0 \text{ s.t. } 0 \le f(n) \le c\,g(n) \;\; \forall n \ge n_0\}.$$



~~Write $f(n) = O(g(n))$, e.g. $2n^2 = O(n^3)$.~~
- ~~$=$ is funny one-way equality~~
- ~~n^2, n^3 are functions, not values but writing carefully (e.g., $\lambda n. n^2$) is tedious.~~

O-notation is a little sloppy, but convenient.
But remember that $O(g(n))$ is really a set of functions.

What about when used in an expression?
$f(n) = n^2 + O(n)$ means
$f(n) = n^2 + h(n)$ for some $h(n) \in O(n)$.
$f(n) \leq n^2 + cn \quad \forall n \geq n_0$ for some $n_0, c > 0$

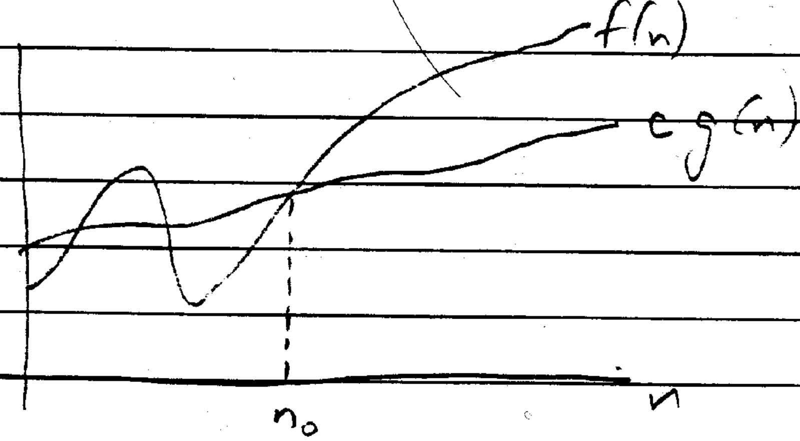
O-notation gives upper bound only.

Ω-notation

For lower bounds.

$$\Omega(g(n)) = \{ f(n) : \exists \text{ consts } c, n_0 > 0 \text{ s.t. } 0 \leq c\,g(n) \leq f(n) \ \forall n \geq n_0 \}$$

CS 325-X94 L2 P3

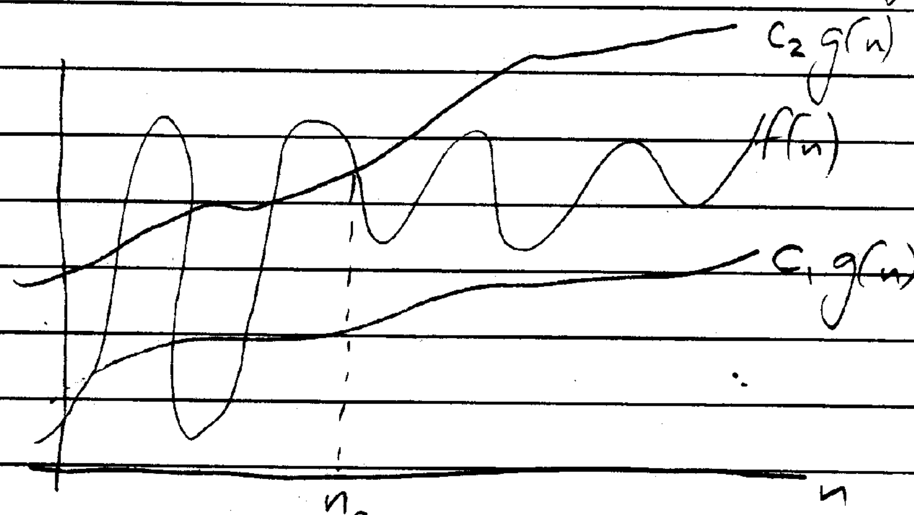


Same equality conventions, e.g., $n^2 = \Omega(n \lg n)$

Θ-notation

For tight bounds — to within a const factor.

$$\Theta(g(n)) = \{ f(n) : \exists \text{ consts } c_1, c_2, n_0 > 0 \text{ s.t. } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \ \forall n \ge n_0 \}$$



Same equality conventions, e.g.,

$\frac{1}{2}n^2 - 2n = \Theta(n^2)$

Proof: $c_1 = \frac{1}{4}$, $c_2 = \frac{1}{2}$, $n_0 = 8$.

$\frac{1}{2}n^2 - 2n \geq \frac{1}{4}n^2$ $\quad$ $\frac{1}{2}n^2 - 2n \underset{0 \leq 2n}{\leq} \frac{1}{2}n^2$

$\frac{1}{4}n^2 \geq 2n$ $\quad$ true $\forall n \geq 0$.

$n \geq 8$

Thm: Leading constants and low-order (additive) terms don't matter.

Justification (not proof):

Can choose consts high enough to make high-order terms swamp other terms.

↳ e.g. $1{,}000{,}000\, n + n^2 \leq 2n^2 \;\; \forall n \geq 1{,}000{,}000$

so, $n_0 = 1{,}000{,}000$, $c = 2$

* Thm: $f(n) = \Theta(g(n))$ iff $(f(n) = O(g(n))$ and $f(n) = \Omega(g(n)))$

Other asymptotic notation

$o(g(n)) = \{f(n) : \forall$ const $c > 0$, $\exists$ const $n_0 > 0$ s.t. $0 \leq f(n) < c\,g(n)\ \forall n \geq n_0\}$

i.e. no matter how small

i.e. possibly quite large

Idea: $f(n)$ becomes insignificant w.r.t. $g(n)$ as $n \to \infty$, i.e.,

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

no matter how big

possibly quite large

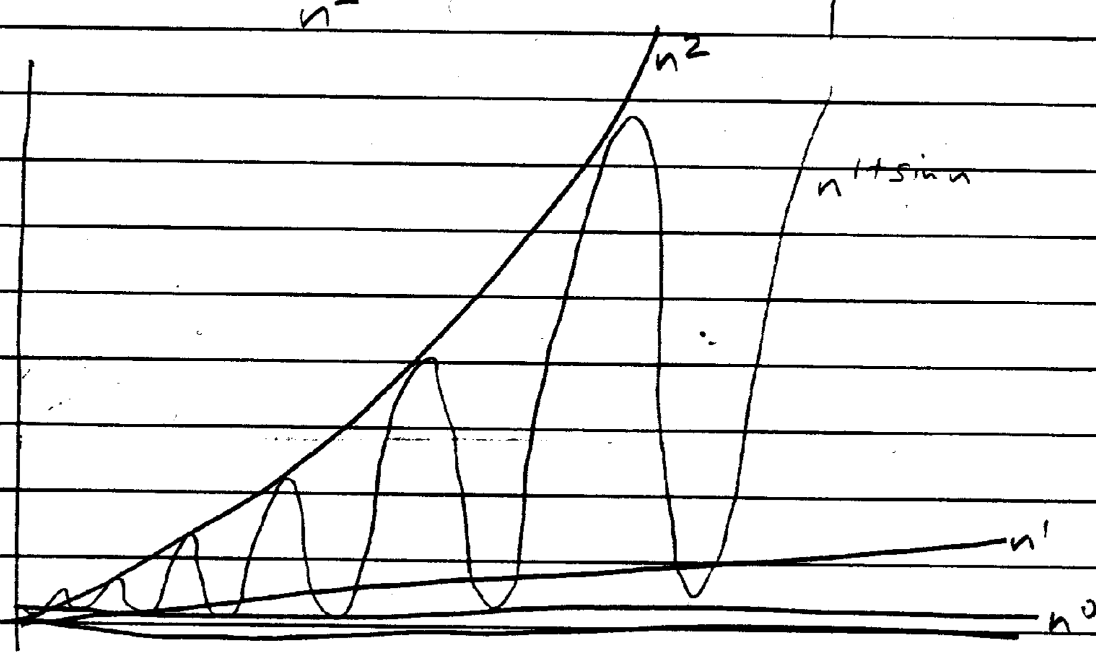
* $\omega(g(n)) = \{ f(n) : \forall c > 0, \exists n_0 > 0 \ni 0 \leq c\,g(n) < f(n) \;\; \forall n \geq n_0$

i.e. $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \}$

Similarly, $f(n) = \omega(g(n)) \implies$
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

O is like $\leq$
Ω $\geq$
Θ $=$
o $<$
ω $>$

But, for reals a, b, exactly one of
$a < b$, $a = b$, $a > b$ holds.
Not nec. true for functions.
Example: n and $n^{1+\sin n}$ can't be
compared using the above notations.
$n^{1+\sin n} = n^0 = 1$ when $\sin n = -1$
n 0
n^2 1



Notation	Name	Intuition	As $n \to \infty$, eventually...	Definition
$f(n) \in O(g(n))$	Big Omicron; Big O; Big Oh	f is bounded above by g (up to constant factor) asymptotically	$f(n) \leq g(n) \cdot k$	$\exists (k > 0), n_0 : \forall (n > n_0)\ \lvert f(n) \rvert \leq \lvert g(n) \cdot k \rvert$ or $\exists (k > 0), n_0 : \forall (n > n_0)\ f(n) \leq g(n) \cdot k$
$f(n) \in \Omega(g(n))$	Big Omega	f is bounded below by g (up to constant factor) asymptotically	$\lvert f(n) \rvert \geq g(n) \cdot k$	$\exists (k > 0), n_0 : \forall (n > n_0)\ \lvert g(n) \cdot k \rvert \leq \lvert f(n) \rvert$
$f(n) \in \Theta(g(n))$	Big Theta	f is bounded both above and below by g asymptotically	$g(n) \cdot k_1 \leq f(n) \leq g(n) \cdot k_2$	$\exists (k_1, k_2 > 0), n_0 : \forall (n > n_0)\ \lvert g(n) \cdot k_1 \rvert < \lvert f(n) \rvert < \lvert g(n) \cdot k_2 \rvert$
$f(n) \in o(g(n))$	Small Omicron; Small O; Small Oh	f is dominated by g asymptotically	$f(n) < g(n) \cdot k$	$\forall (k > 0), \exists n_0 : \forall (n > n_0)\ \lvert f(n) \rvert < \lvert g(n) \cdot k \rvert$
$f(n) \in \omega(g(n))$	Small Omega	f dominates g asymptotically	$f(n) > g(n) \cdot k$	$\forall (k > 0), \exists n_0 : \forall (n > n_0)\ \lvert g(n) \cdot k \rvert < \lvert f(n) \rvert$
$f(n) \sim g(n)$	on the order of	f is equal to g asymptotically	$\lvert f(n) - g(n) \cdot k \rvert < \varepsilon$	$\lim_{n \to \infty} \frac{f(n)}{g(n)} = k, 0 < k < \infty$

Orders of Growth

Ten Orders of Growth

Let's assume that your computer can perform 10,000 operations (e.g., data structure manipulations, database inserts, etc.) per second. Given algorithms that require lg n, $n^{½}$, n, n^2, n^3, n^4, n^6, 2^n, and $n!$ operations to perform a given task on n items, here's how long it would take to process 10, 50, 100 and 1,000 items.

	n			
	10	**50**	**100**	**1,000**
lg n	0.0003 sec	0.0006 sec	0.0007 sec	0.0010 sec
$n^{½}$	0.0003 sec	0.0007 sec	0.0010 sec	0.0032 sec
n	0.0010 sec	0.0050 sec	0.0100 sec	0.1000 sec
n lg n	0.0033 sec	0.0282 sec	0.0664 sec	0.9966 sec
n^2	0.0100 sec	0.2500 sec	1.0000 sec	100.00 sec
n^3	0.1000 sec	12.500 sec	100.00 sec	1.1574 day
n^4	1.0000 sec	10.427 min	2.7778 hrs	3.1710 yrs
n^6	1.6667 min	18.102 day	3.1710 yrs	3171.0 cen
2^n	0.1024 sec	35.702 cen	4×10^{16} cen	1×10^{166} cen
$n!$	362.88 sec	1×10^{51} cen	3×10^{144} cen	1×10^{2554} cen

Table 1: Time required to process n items at a speed of 10,000 operations/sec using eight different algorithms.

Note: The units above are seconds (sec), minutes (min), hours (hrs), days (day), and centuries (cen)!

The Explosive Growth of 2^n

n						
15	**20**	**25**	**30**	**35**	**40**	**45**
3.28 sec	1.75 min	55.9 min	1.24 days	39.8 days	3.48 yrs	1.12 cen

Table 2: Time required to process n items at a speed of 10,000 operations/sec using a 2^n algorithm.

The Explosive Growth of $n!$

n						
11	**12**	**13**	**14**	**15**	**16**	**17**
1.11 hrs	13.3 hrs	7.20 days	101 days	4.15 yrs	66.3 yrs	11.3 cen

Table 3: Time required to process *n* items at a speed of 10,000 operations/sec using an *n*! algorithm.