CS61B Lecture #16: Complexity

What Are the Questions?

ncipal concern throughout engineering:

eer is someone who can do for a dime what any fool
r a dollar."

an

al cost (for programs, time to run, space requirements).

ent costs: How much engineering time? When delivered?

nce costs: Upgrades, bug fixes.

failure: How robust? How safe?

ram *fast enough*? Depends on:

purpose;

input data.

pace (memory, disk space)?

ends on what input data.

cale, as input gets big?

Enlightening Example

a text corpus (say 10^9 bytes or so), and find and print
quently used words, together with counts of how often

nuth): Heavy-Duty data structures

implementation, randomized placement, pointers galore,
ges long.

oug McIlroy): UNIX shell script:

’[:alpha:]’ ’[\n*]’ < *FILE* | \

\
r -k 1,1 | \

ter?

h faster,

ok 5 minutes to write and processes 1GB in ≈ 256 sec.

cases, almost anything will do: Keep It Simple.

Cost Measures (Time)

r execution time

o this at home:

`e java FindPrimes 1000`

es: easy to measure, meaning is obvious.

te where time is critical (real-time systems, e.g.).

ages: applies only to specific data set, compiler, machine,

tement counts of # of times statements are executed:

es: more general (not sensitive to speed of machine).

ages: doesn't tell you actual time, still applies only to
ata sets.

ecution times:

ormulas for execution times as functions of input size.

es: applies to all inputs, makes scaling clear.

age: practical formula must be approximate, may tell
about actual time.

Asymptotic Cost

ecution time lets us see *shape* of the cost function.

e approximating anyway, pointless to be precise about
s:

on small inputs:

ays pre-calculate some results.

or small inputs not usually important.

more interested in *asymptotic behavior* as input size
s very large.

factors (as in "off by factor of 2"):

anging machines causes constant-factor change.

ract away from (i.e., ignore) these things?

Handy Tool: Order Notation

try to produce specific functions that specify size, but
ies of functions with similarly behaved magnitudes.

ething like "f is bounded by g if it is in g's family."

tion $g(x)$, the functions $2g(x)$, $0.5g(x)$, or for any $K > 0$,
ave the same "shape". So put all of them into g's family.

$h(x)$ such that $h(x) = K \cdot g(x)$ for $x > M$ (for some
has g's shape "except for small values." So put all of
family.

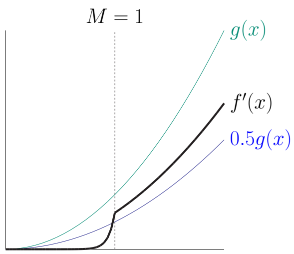
its, throw in all functions whose absolute value is everywhere
ber of g's family. Call this set $O(g)$ or $O(g(n))$.

r limits, throw in all functions whose absolute value is
$\geq$ some member of g's family. Call this set $\Omega(g)$.

e $\Theta(g) = O(g) \cap \Omega(g)$—the set of functions *bracketed*
by two members of g's family.

Big Omega

y bounding from below:



$\frac{1}{2}g(x)$ as long as $x > 1$,

g's "bounded-below family," written

$$f'(x) \in \Omega(g(x)),$$

gh $f(x) < g(x)$ everywhere.

de: Various Mathematical Pedantry

if I am going to talk about $O(\cdot)$, $\Omega(\cdot)$ and $\Theta(\cdot)$ as sets of really should write, for example,

$f \in O(g)$ instead of $f(x) \in O(g(x))$

$(x) \in O(g(x))$ is short for $\lambda\, x.\ f(x) \in O(\lambda\, x.\ g(x))$.

d notation outside this course, in fact, is $f(x) = O(g(x))$, ly, I think that's a serious abuse of notation.

Why It Matters

ientists often talk as if constant factors didn't matter he difference of $\Theta(N)$ vs. $\Theta(N^2)$.

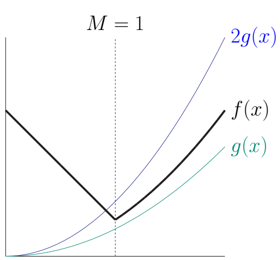
ey do matter, but at some point, constants always get

	$\sqrt{n}$	n	$n \lg n$	n^2	n^3	2^n
	1.4	2	2	4	8	4
	2	4	8	16	64	16
	2.8	8	24	64	512	256
	4	16	64	256	4,096	65,636
	5.7	32	160	1024	32,768	4.2×10^9
	8	64	384	4,096	262,144	1.8×10^{19}
	11	128	896	16,384	2.1×10^9	3.4×10^{38}
	⋮	⋮	⋮	⋮	⋮	⋮
	32	1,024	10,240	1.0×10^6	1.1×10^9	1.8×10^{308}
	⋮	⋮	⋮	⋮	⋮	⋮
	1024	1.0×10^6	2.1×10^7	1.1×10^{12}	1.2×10^{18}	$6.7 \times 10^{315,652}$

: replace column n^2 with $10^6 \cdot n^2$ and it still becomes y 2^n.

Big Oh

y bounding from above.



$2g(x)$ as long as $x > 1$,

g's "bounded-above family," written

$$f(x) \in O(g(x)),$$

gh (in this case) $f(x) > g(x)$ everywhere.

Big Theta

previous slides, we not only have $f(x) \in O(g(x))$ and $)),\ldots$

$(x) \in \Omega(g(x))$ and $f'(x) \in O(g(x))$.

marize this all by saying $f(x) \in \Theta(g(x))$ and $f'(x) \in$

How We Use Order Notation

mathematics, you'll see $O(\ldots)$, etc., used generally to ds on functions.

,

$$\pi(N) = \Theta(\frac{N}{\ln N})$$

d prefer to write

$$\pi(N) \in \Theta(\frac{N}{\ln N})$$

is the number of primes less than or equal to N.)

ee things like

$= x^3 + x^2 + O(x)$ (or $f(x) \in x^3 + x^2 + O(x)$),

$f(x) = x^3 + x^2 + g(x)$ where $g(x) \in O(x)$.

poses, the functions we will be bounding will be *cost* unctions that measure the amount of execution time or of space required by a program or algorithm.

Using the Notation

order notation for any kind of real-valued function.

hem to describe cost functions. Example:

```
 position of X in list L, or -1 if not found. */
(List L, Object X) {

: = 0; L != null; L = L.next, c += 1)
 (X.equals(L.head)) return c;
n -1;
```

esentative operation: number of `.equals` tests.

th of L, then loop does *at most* N tests: *worst-case* sts.

al # of instructions executed is roughly proportional worst case, so can also say worst-case time is $O(N)$, f units used to measure.

provision (in defn. of $O(\cdot)$) to ignore empty list.

Effect of Nested Loops

s often lead to polynomial bounds:

```
 i = 0; i < A.length; i += 1)
int j = 0; j < A.length; j += 1)
 (i != j && A[i] == A[j])
 return true;
alse;
```

: is $O(N^2)$, where $N =$ `A.length`. *Worst-case time* is

icient though:

```
 i = 0; i < A.length; i += 1)
int j = i+1; j < A.length; j += 1)
 (A[i] == A[j]) return true;
alse;
```

ase time is proportional to

$$- 1 + N - 2 + \ldots + 1 = N(N-1)/2 \in \Theta(N^2)$$

ic time unchanged by the constant-factor speed-up).

Binary Search: Slow Growth

```
 is an element of S[L .. U]. Assumes
ding order, 0 <= L <= U-1 < S.length. */
String X, String[] S, int L, int U) {
eturn false;
)/2;
 X.compareTo(S[M]);
 0) return isIn(X, S, L, M-1);
ect > 0) return isIn(X, S, M+1, U);
 true;
```

case time, $C(D)$, (as measured by # of calls to `.compareTo`), ize $D = U - L + 1$.

S[M] from consideration each time and look at half the $D = 2^k - 1$ for simplicity, so:

$$\begin{aligned} C(D) &= \begin{cases} 0, & \text{if } D \le 0, \\ 1 + C((D-1)/2), & \text{if } D > 0. \end{cases} \\ &= \underbrace{1 + 1 + \ldots + 1}_{k} + 0 \\ &= k = \lg(D+1) \in \Theta(\lg D) \end{aligned}$$

ne Intuition on Meaning of Growth

oblem can you solve in a given time?

wing table, left column shows time in microseconds to problem as a function of problem size N.

the *size of problem* that can be solved in a second, (31 days), and century, for various relationships between d and problem size.

size.

c) for ze N	Max N Possible in 1 second	1 hour	1 month	1 century
	10^{300000}	$10^{1000000000}$	$10^{8\cdot 10^{11}}$	$10^{10^{14}}$
	10^6	$3.6\cdot 10^9$	$2.7\cdot 10^{12}$	$3.2\cdot 10^{15}$
N	63000	$1.3\cdot 10^8$	$7.4\cdot 10^{10}$	$6.9\cdot 10^{13}$
	1000	60000	$1.6\cdot 10^6$	$5.6\cdot 10^7$
	100	1500	14000	150000
	20	32	41	51

Be Careful

that the worst-case time is $O(N^2)$, since $N \in O(N^2)$ bounds are loose.

ase time is $\Omega(N)$, since $N \in \Omega(N)$, but that does *not* e loop *always* takes time N, or even $K \cdot N$ for some K.

are just saying something about the *function* that maps *argest possible* time required to process any array of

ch as possible about our worst-case time, we should try ound: in this case, we can: $\Theta(N)$.

hat still tells us nothing about *best-case* time, which n we find X at the beginning of the loop. Best-case time

rsion and Recurrences: Fast Growth

of recursion. In the worst case, both recursive calls

```
ff X is a substring of S */
curs(String S, String X) {
uals(X)) return true;
ngth() <= X.length()) return false;

(S.substring(1), X) ||
(S.substring(0, S.length()-1), X);
```

) to be the worst-case cost of `occurs(S,X)` for S of f fixed size N_0, measured in # of calls to `occurs`. Then

$$C(N) = \begin{cases} 1, & \text{if } N \le N_0, \\ 2C(N-1)+1 & \text{if } N > N_0 \end{cases}$$

ws exponentially:

$$N-1) + 1 = 2(2C(N-2)+1) + 1 = \ldots = \underbrace{2(\cdots 2}_{N-N_0} \cdot 1 + 1) + \ldots + 1$$

$$^{N_0} + 2^{N-N_0-1} + 2^{N-N_0-2} + \ldots + 1 = 2^{N-N_0+1} - 1 \in \Theta(2^N)$$

...other Typical Pattern: Merge Sort

```
... L) {
...() < 2) return L;
... and L1 of about equal size;
...)); L1 = sort(L1);
...e of L0 and L1
```

Merge ("combine into a single ordered list") takes time proportional to size of its result.

...at size of L is $N = 2^k$, worst-case cost function, $C(N)$,
... merge time (which is proportional to # items merged):

$$C(N) = \begin{cases} 0, & \text{if } N < 2; \\ 2C(N/2) + N, & \text{if } N \geq 2. \end{cases}$$
$$= 2(2C(N/4) + N/2) + N$$
$$= 4C(N/4) + N + N$$
$$= 8C(N/8) + N + N + N$$
$$= N \cdot 0 + \underbrace{N + N + \ldots + N}_{k = \lg N}$$
$$= N \lg N$$

...an say it's $\Theta(N \lg N)$ for arbitrary N (not just 2^k).