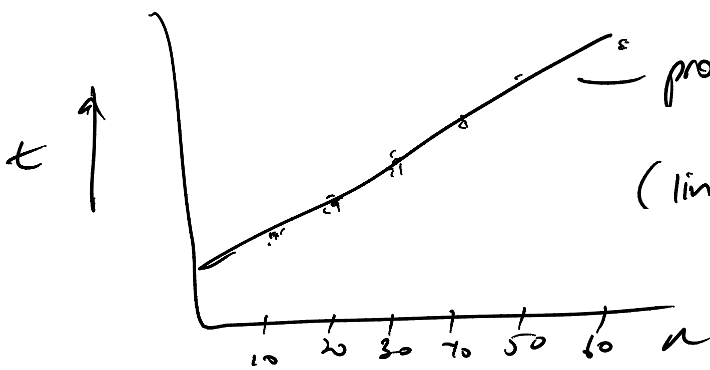
Algorithm Analysis (Chp. 2)

- How do we compare the running time of algorithms?
  - the empirical approach:
    - ⟶ experiments, observation
    - use your Timer class, place start(), stop() around code fragments (e.g. sort)
    - collect running time of a piece of code
      - will running it once suffice? No
      - why not?
        - diff'rt platforms (CPUs, memory, OS)
        - # of tasks running on CPU (Linux is multitasking OS)
        - need statistics honestly

Thursday, September 28, 2006
3:28 PM

```
int sum(int n)
{ int partialSum;
  Timer t;
  t.start();
  partialSum = 0;
  for(int i=1; i<=n; i++) partialSum += i*i*i;
  t.end();
  cout << "took " << t.elapsed_ms() << " ms" << endl;
  return partialSum;
}
```

Run it several times, changing n at random; plot



probably a line
(linear complexity)

Thursday, September 28, 2006
3:28 PM

- empirical approach is nice, but we'd like to be able to predict running time
- can do this by counting # execution steps of alg.

operation counts

```
int sum (int n)
{
    int ps;
    for (int i=1; i<=n; i++)
        ps += i*i*i;
    return ps
}
```

$\emptyset$ — int ps;

$2N+2$ — for ($\underbrace{\text{int } i=1;}_{1}$ $\underbrace{i<=n;}_{N+1}$ $\underbrace{i{+}{+}}_{N}$)

$4N$ — ps += i*i*i; (2 *, 1 +, 1 =)

1 — return ps

$0+1+(2N+2)+(4N+1) = 6N+4$

\# operations, \# statements
- declarations = 0

- Basic rules:
  1. for loops: # statements inside × iterations
  2. nested for loops:
     - analyze inside-out
     - product of all loops

$$
n\left[\begin{array}{l} \text{for}(i=0;\ i<n,\ i{+}{+}) \\ n\left[\begin{array}{l}\text{for}(j=0;\ j<n;\ j{+}{+}) \\ \quad k{+}{+};\end{array}\right.\end{array}\right.
$$

$$n \times n = n^2$$

  3. Consecutive statements: just add up
  4. conditionals: use larger of the branches
  5. recursion: the tricky one
     - we'll come back to that one

- given (or actual #) estimate of # statements, come up with asymptotic function to describe order (magnitude) of statements,

e.g. $T(N)$: $6N+4 \approx O(N)$ ← big-oh : "order N"

same class of function in asymptotic sense ~ only differs in some constant amount.

- more formally
$T(n) = O(f(N))$ if $\exists$ c, n_0 s.t. $T(N) \leq c f(N)$ where $N \geq n_0$

let's say $T(N) = N+1$

$N+1 \leq cN$ where $N \geq n_0$ if we let $n_0 = 100$, $c = 2$

✓ $100+1 \leq 200$

$N+1 \in O(N)$

basically, we can drop the constants

- in general, we can drop the constants, & lower order terms

e.g. $T(N) = O(\cancel{2}N^2 + N) = O(N^2 + \cancel{N}) = O(N^2)$

- big-oh $O(f(N))$ specifies <u>upper bound</u> — practically speaking, <u>worst-case</u> running time of alg

- other important analytical definitions:

$\Omega(g(N))$

$T(N) = \Omega(g(N))$ if $\exists c, n_0 \mid T(N) \geq c\,g(N)$

means that $g(N)$ is the alg's lower bound —

alg must run at least as fast as this

(best case analysis)

$T(n) = \Theta(h(N))$ iff $T(N) = O(h(N)) = \Omega(h(N))$

this is used to denote <u>average</u> running time

finally,

little-oh

$T(N) = o(p(N))$ iff $\forall c \; \exists n_0 \mid T(N) < c\,p(N)$ when $n > n_0$

big-oh allows $\leq$ equality

- what's the big idea? want to $\boxed{\text{compare}}$ relative running times (bounds) of alg's.

(empirical approach: sort of an absolute metric — here we want relative & predictive)

- relative orders:

$O(1) < O(\lg N) < O(\lg^2 N) < O(N) < O(N \lg N)$

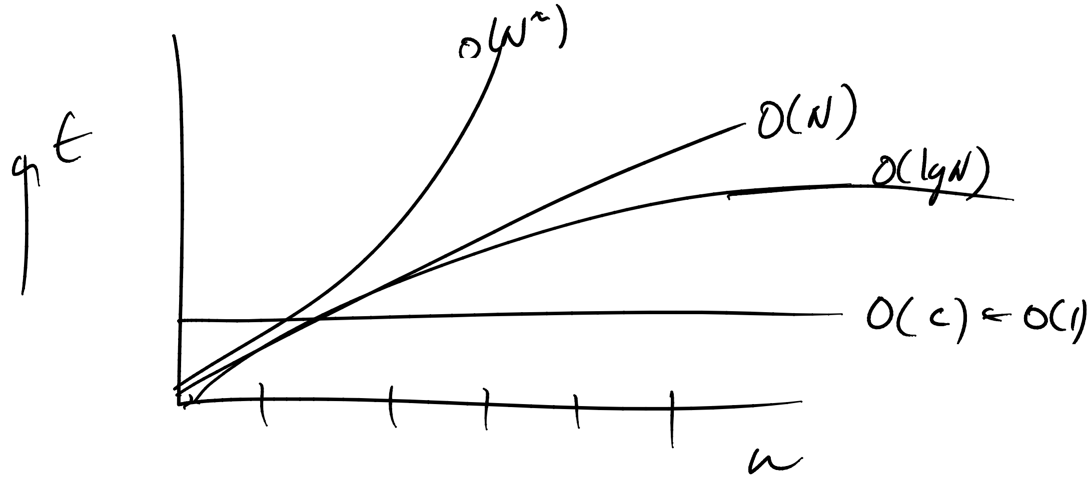
constant, logarithmic, iterated logarithmic, linear, loglinear

$< O(N^2) < O(N^3) < O(2^N)$

quadratic, cubic, exponential

each $O(f(n))$ describes growth rate

Thursday, September 28, 2006
3:28 PM



5. recursion: the tricky one

— involves solving recurrence relations

RECURRENCE

(or proving, e.g. by induction)

e.g.
```
long fib(int n)
{
    if (n <= 1) return 1
    else return fib(n-1) + fib(n-2)
}
```

$T(0) = T(1) = 1$

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

How to solve?

- for $n > 4$, $T(N) \geq \left(\frac{3}{2}\right)^n$ exponential
- prove by induction

$T(n) = T(n-1) + T(n-2) \geq \left(\frac{3}{2}\right)^n$

assume it holds for $n > 4$

show $T(N+1) \geq \left(\frac{3}{2}\right)^{n+1}$

from

$$T(n+1) = T(n+1-1) + T(n+1-2)$$
$$= T(n) + T(n-1)$$
$$\left(\frac{3}{2}\right)^n + \left(\frac{3}{2}\right)^{n-1}$$

want to get it in terms of $\left(\frac{3}{2}\right)^{n+1}$ so we get this

$$\left(\frac{3}{2}\right)^n + \left(\frac{3}{2}\right)^{n-1}$$
$$= \left(\frac{3}{2}\right)^{-1}\left(\frac{3}{2}\right)^{n+1} + \left(\frac{3}{2}\right)^{-1}\left(\frac{3}{2}\right)^{-1}\left(\frac{3}{2}\right)^{n+1}$$

$$= \left(\frac{2}{3}\right)\left(\frac{3}{2}\right)^{n+1} + \left(\frac{2}{3}\right)^2\left(\frac{3}{2}\right)^{n+1}$$
$$= \left(\frac{2}{3} + \frac{4}{9}\right)\left(\frac{3}{2}\right)^{n+1} = \left(\frac{10}{9}\right)\left(\frac{3}{2}\right)^{n+1} \geq \left(\frac{3}{2}\right)^{n+1}$$

QED