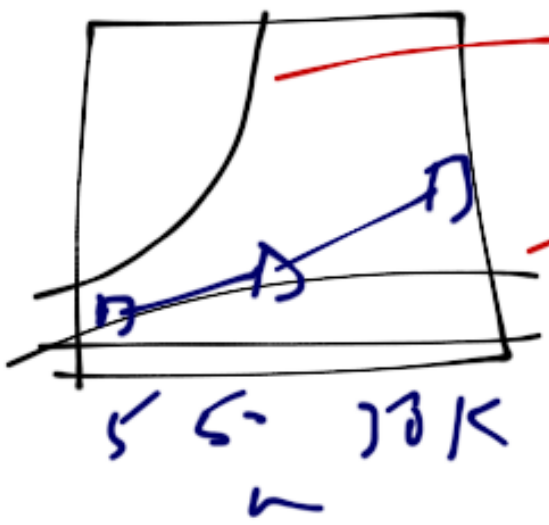
Alg. Analysis (Chp. 2)

- goal: compare & predict alg. performance (run time)
- empirical approach:
  - use timer_t class



- graph & report (e.g. in tables)

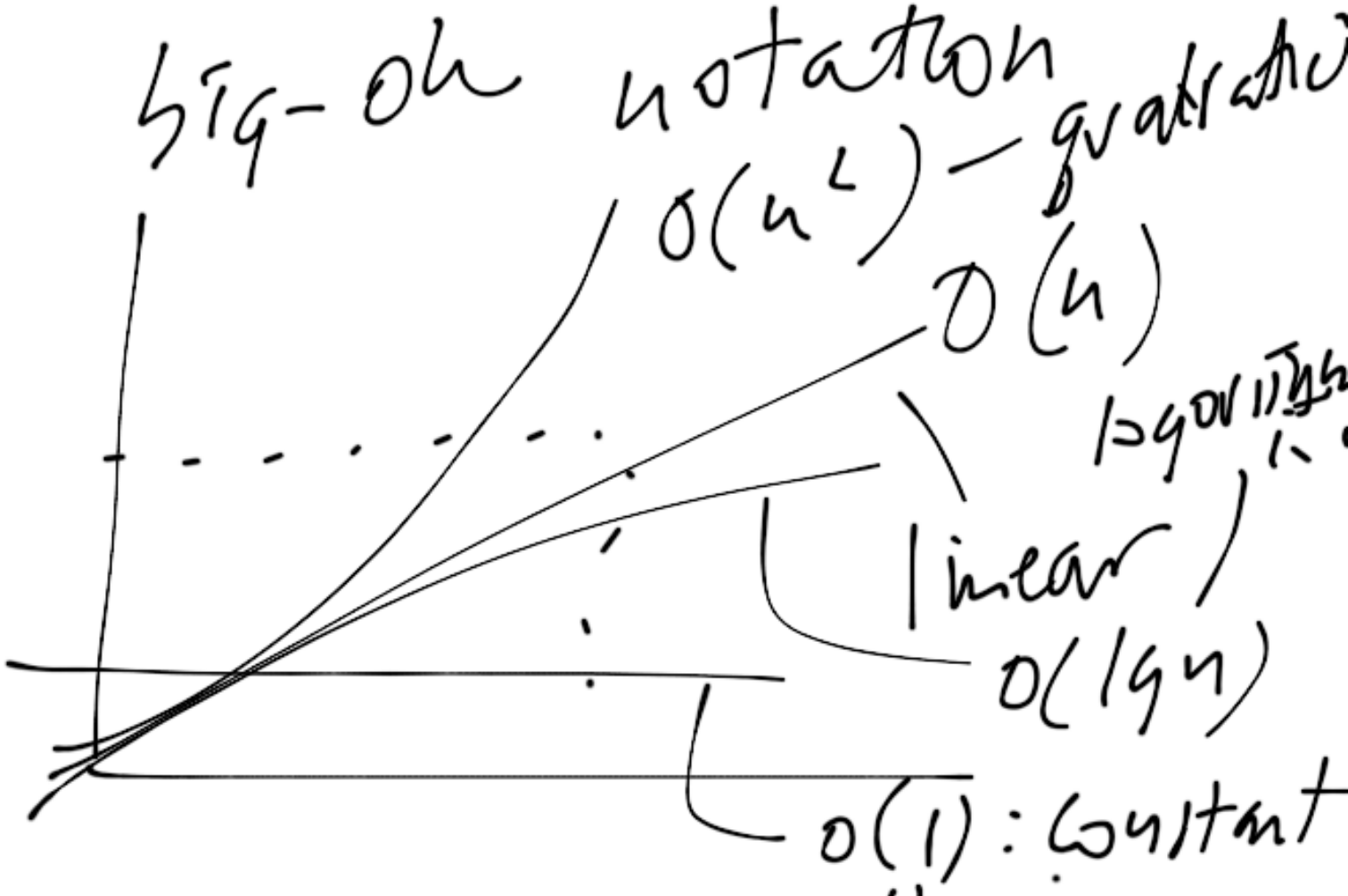
(still relevant today e.g. graphics papers)

- problems (w/ empirical approach):
  - machine-specific
  - unreliable for few runs — need to run several times

- any given run of Unix may take a diff. amount of time; Unix is multi-tasking OS: many processes running "simultaneously" but only one CPU
⇒ context switching

- analytical approach :

big-oh notation



n : input size

(e.g. n : size of array,

5, 50, 500, 50,000)

Comparisons:

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

constant logarithmic iterated

$< O(n)$ linear logarithm

$< O(n \lg n) < O(n^2)$

log linear quadratic

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

cubic exponential

- What to count?

operation
(cost)

```
int sum(int n)
{
  int sum;          0 ——————
  sum = 0;          1 ——————
  for (i=1 to n)    2n+2 ——————
  sum += i*i*i;     2/n ——————
  return sum;       1 ——————
}
```

6n+4

- $6n + 4$:

  - reduce all constants to 1

    $\Rightarrow 1n + 1 \Rightarrow n + 1$

- apply $O(f(n))$ definition

$T(n) = O(f(n))$ if

$\exists\ c, n_0 \mid T(n) \leq c f(n)$

when $n \geq n_0$.

- $n+1 \in O(n)$ ✓

$T(n)$ $O(f(n))$

$f(n)$ is n

$n+1 \leq cn$

($T(n) \leq cf(n)$

↑

such that)

when $n \geq n_0$

e.g., let $n_0 = 100$, $c = 2$

$100+1 \leq 2 * 100$?

- in general, drop lower-order terms

$$T(n) = O(2n^2)$$
$$= O(n^2 + n)$$
$$= O(n^2)$$

- big-oh ($O(f(n))$)
  specifies upper bound:
  alg can't be any worse than $O(f(n))$

- $O(f(n))$: worst-case running time

- $\Omega(g(n))$:

$T(n) = \Omega(g(n))$ if

$\exists\, c, n_0$ |

$T(n) \geq c\, g(n)$

- $\Omega(g(n))$ means that $g(n)$ must be the alg's lower bound

alg. is at least as fast as this ($\Omega(g(n))$)
(slow)

(best-case performance)

- $T(n) = \Theta(h(n))$ iff

$T(n) = O(h(n))$ &

$T(n) = \Omega(h(n))$

$\Omega(n) = T(n) = O(n)$

$\Rightarrow$ upper & lower bounds the same, so $\Theta()$ denotes average running time.

- $T(n) = o(p(n))$ if

little oh

$\forall c \; \exists n_0$ |

$T(n) < cp(n)$ when

$n > n_0$

strict inequality

(cf. $O(f(n))$ allows $\leq$

little oh does not)

(no one uses little-oh :)

- program analysis
(rules of thumb)

1. for loops:
   - no. statements x
     no. iterations

2. nested loops:
   - count inside out

O(n) — for(i=0; i<n; i++)

O(n) — for(j=0; j<n; j++)
        k++;

O(n²)

3. Consecutive statements:
just add up

(Σ)

4. Conditionals:
use larger of the
branches
(overestimate rather
than underestimate)

5. recursion: the tricky one

— need to solve recurrence relations

eg- fib(int n)

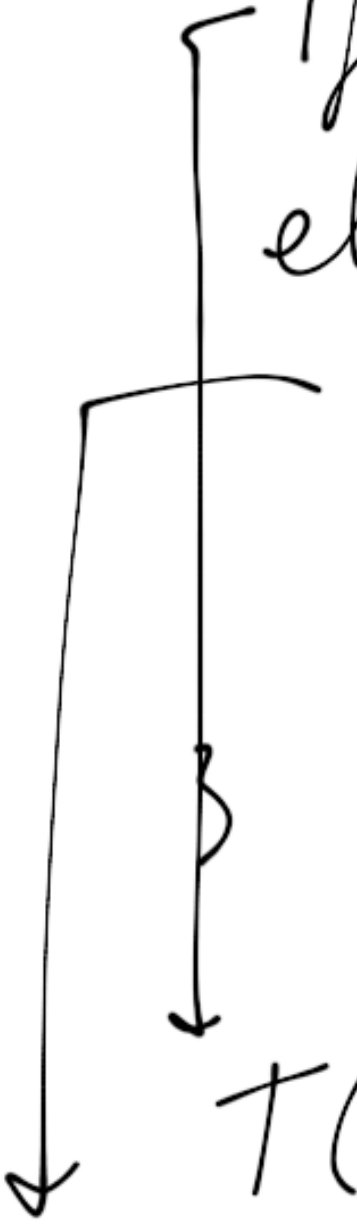
{ if (n<=1) return 1;

else

return fib(n-1) +

fib(n-2)

}



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

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

book says for $n > 4$,

$$T(n) \geq \left(\frac{3}{2}\right)^n$$

inductive proof:

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

assume this holds for $n > 4$

show it holds for $n+1$

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

so if n is $n+1$,

$(n+1)-1 = n$

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

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

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

(from top $T(n) \geq \left(\frac{3}{2}\right)^n$,

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

$$T(n) \geq \left(\frac{3}{2}\right)^n$$

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

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

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

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

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

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

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

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

which is larger than $\left(\frac{3}{2}\right)^{n+1}$

■

- generally we deal with recursive calls that split lists up in two, then recurse on each half

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$T(n)$: which $O(f(n))$ class does this fall?

$2T\left(\frac{n}{2}\right)$: time to process half the list, twice

$+ n$