6.006
Spring 2008
L2

Asymptotic notation

Doc distance summary

Merge-sort
- Divide & Conquer
- Analysis of recurrences

Handouts

Doc dist V5 & V6
PS1

Readings: CLRS Chapter 4

Asymptotics

Parameterize input size as n (m, t, etc)

Many different inputs of size n

$T(n)$ = worst case running time for input size n

$= \max_{X: \text{ input of size } n}$ running time on X

How can we be precise?

Don't care about $T(n)$ for small n

" " constant factors (diff computers, languages..)

Suppose $T(n) = 4n^2 + 22n - 12$ μs.

only care about highest order term, without constant

Say $T(n)$ is $O(g(n))$ if $\exists n_0, \exists c$ s.t.

$$0 \leq T(n) \leq c \cdot g(n) \quad \text{for all } n \geq n_0$$

$$0 \leq 4n^2 + 22n - 12 \leq 26n^2 \quad \text{for } n \geq 1$$

$$\therefore \quad 4n^2 + 22n - 12 = O(n^2)$$

read as "is" or $\in$ belongs to a set

O : upper bound

Say $T(n) = \Omega(g(n))$ if $\exists n_0, \exists d$ s.t.

$$0 \leq d \cdot g(n) \leq T(n) \quad \text{for all } n \geq n_0$$

$$T(n) = 4n^2 + 22n - 12 \geq n^2 \quad \text{for } n \geq 1$$

$$\therefore \quad T(n) = \Omega(n^2)$$

Say $T(n) = \Theta(g(n))$ iff $T(n) = O(g(n))$ and $T(n) = \Omega(g(n))$

Ω : lower bound

Θ : high order term is $g(n)$

Doc dist review

×2
- Read file
- Make word list — + op on list — $\Theta(n^2)$
- count frequencies — double loop — $\Theta(n^2)$
- Sort in order — insertion sort, double loop — $\Theta(n^2)$

Compute angle $= \arccos\left(\frac{D_1 \cdot D_2}{\|D_1\| \cdot \|D_2\|}\right)$ — $\Theta(n)$

Optimizations

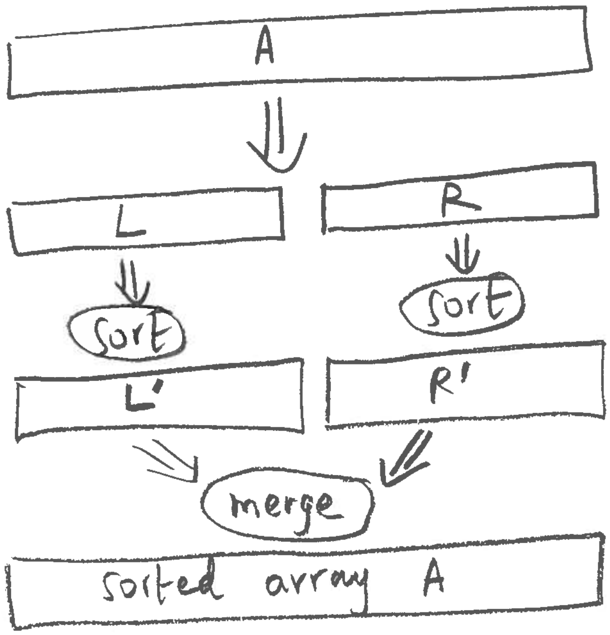
		Time	Bobsey vs. Lewis
V1	initial	?	
V2	add profiling	195 s	
V3	wordlist.extend(..)	84 s	$\Theta(n^2) \to \Theta(n)$
V4	dictionaries in count-freq	41 s	$\Theta(n^2) \to \Theta(n)$
V5	process words rather than chars in get words from string	13 s	$\Theta(n) \to \Theta(n)$
V6	merge-sort rather than insertion sort	6 s	$\Theta(n^2) \to \Theta(n \lg n)$

$\lg = \log_2$

V6B eliminate sorting altogether to get $\Theta(n)$ algorithm — ~1 s ← courtesy Mason Tang, 6.006 student

Merge - sort

Divide / Conquer / Combine paradigm.



input array of size n

2 arrays of size n/2

2 sorted arrays of size n/2

size n.

---

5 4 7 3 6 1 9 2

3 4 5 7 1 2 6 9

i j

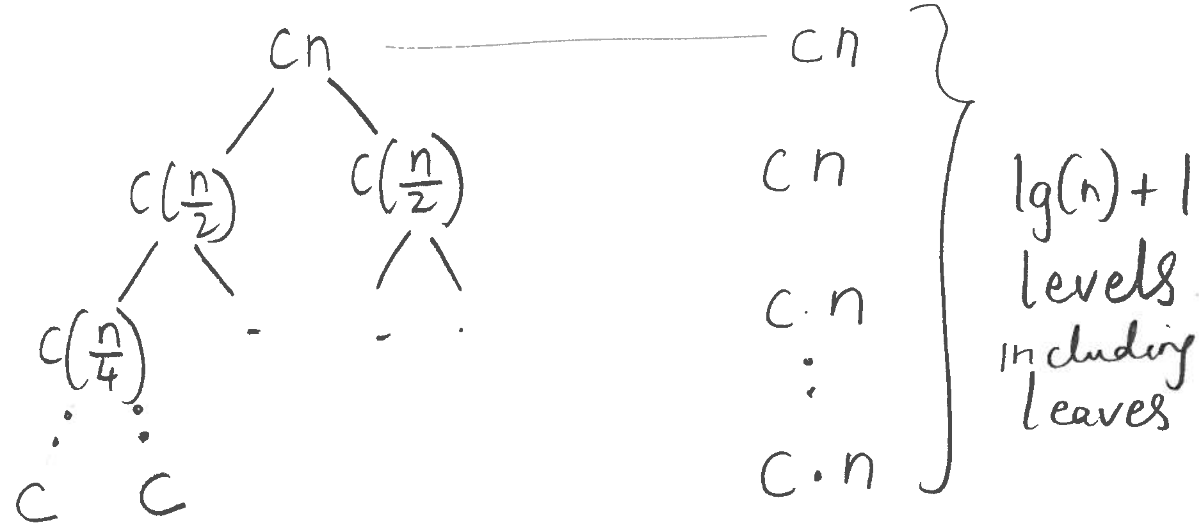
"two finger" algorithm for merge.

1	2	3	4	5	6	7	9
inc j	inc j	inc i	inc i	inc i	inc j	inc i (array L done)	inc j (array R done)

$$T(n) = \underbrace{c_1}_{\text{divide}} + \underbrace{2 \cdot T(n/2)}_{\text{recursion}} + \underbrace{c \cdot n}_{\text{merge}}$$

$$T(n) \cong 2T(n/2) + cn$$ only keep high order terms.

$$= cn + 2\left(c \cdot \left(\frac{n}{2}\right) + 2\left(c \cdot \left(\frac{n}{4}\right) + \cdots\right)\right)$$



$$T(n) = c \cdot n(\lg n + 1) = \theta(n \lg n).$$

Experiment

insertion-sort $\theta(n^2)$

merge-sort $\theta(n \lg n)$ $n = 2^i$

built-in sort $\theta(n \lg n)$?

test_merge routine   merge-sort takes $\approx 2.2\, n \lg n$ μS

test_insert   insertion-sort takes $\approx 0.2\, n^2$ μS

Built-in sort (sorted) takes $\approx 0.1\, n \lg n$ μS

20X constant factor because it is written in C when is merge-sort (in Python) $2n\lg n$

better than insertion sort (in C') $0.01n^2$ ← obtained as 20X over Python insertion-sort

merge-sort wins for $n \geq 2^{12} = 4096$

[better algorithm much more valuable than hardware or compiler even for modest n]

Python cost model: tomorrow's recitation many experiments of this sort. Also PS 1 set ops.