Handouts: mergesort.py

Readings: CLRS Chaps 1-4, 11.1, 11.2
Python cost model
docdist1 ... docdist6

Web: mergesort.py, lecture notes

Admin: HW #1 will be posted today
laptop loaner program
new students?

Outline:
- ☐ Docdist review
- ☐ Asymptotic notation
- ☐ Mergesort:
  - ☐ Divide & Conquer
  - ☐ Code
  - ☐ Analysis / Recurrences
  - ☐ Timing Experiments

Document Distance Review (Bobsey vs. Lewis)

v1	initial	?secs	
v2	profiled	194	$\Theta(n^2)$
v3	concatenate → extend	84	$\Theta(n^2)$
v4	dictionaries instead of lists	41	$\Theta(n^2)$
v5	translate & split	13	$\Theta(n^2)$
v6	merge-sort	6	$\Theta(n \lg n)$
(v7?)	no sorting!	<1	$\Theta(n)$

(Even though sorting is not necessary, it is very worthwhile to look at, so we shall...)

Sorting Problem:

Given a list of n comparable objects, rearrange them into increasing (nondecreasing) order.

6.006
Rivest
L2.2
9/9/08

Input sizes:

Time gets larger as inputs do.
Parameterize size with one or more measures (n, m, ...)
There are many inputs of a given size.

$$T(n) = \text{worst-case running time on an input of size } n$$
$$= \max_{\substack{\text{inputs } x \\ \text{of size } n}} [\text{running time on } x]$$

For insertion sort (ref docdist code, & CLRS §2.1)

$$T(n) \approx \text{const} \cdot n^2 \quad (\text{due to doubly-nested loops})$$

How to be precise about such things?
when * we don't care about T(n) for small n
* " " " " constant factors
(different computers, interpreted/compiled, etc...)

While running time might be
$$4n^2 + 22n - 12 \text{ microseconds}$$
we only care about high-order term ($4n^2$)
but without constant (n^2)
since other terms are negligible (relatively) as n gets large.

"big oh" notation

6.006
Rivest
L2.3
9/9/08

We say
$T(n)$ is $O(g(n))$
if
$\exists n_0$
$\exists c$
s.t. $0 \le T(n) \le c \cdot g(n)$ for all $n \ge n_0$

upper bound

Example: $\underbrace{4n^2+22n-12}$ is $O(n^2)$
since $0 \le$ .. $\le 26n^2$ for $n \ge 1$.
write $4n^2+22n-12 = O(n^2)$ (but not reverse
O always on right)

lower bound

Big Omega:
$T(n) = \Omega(g(n))$
if $(\exists n_0)(\exists c)$ $0 \le c \cdot g(n) \le T(n)$ for all $n \ge n_0$

$4n^2+22n-12 = \Omega(n^2)$ $[c=1, n_0=1]$

both

Big Theta:
$T(n) = \Theta(g(n))$ iff $T(n) = O(g(n))$ & $T(n) = \Omega(g(n))$

$\equiv$ $g(n)$ is high-order term in $T(n)$ (up to constant)

$\therefore$ $T(n) = 4n^2+22n-12 = \Theta(n^2)$

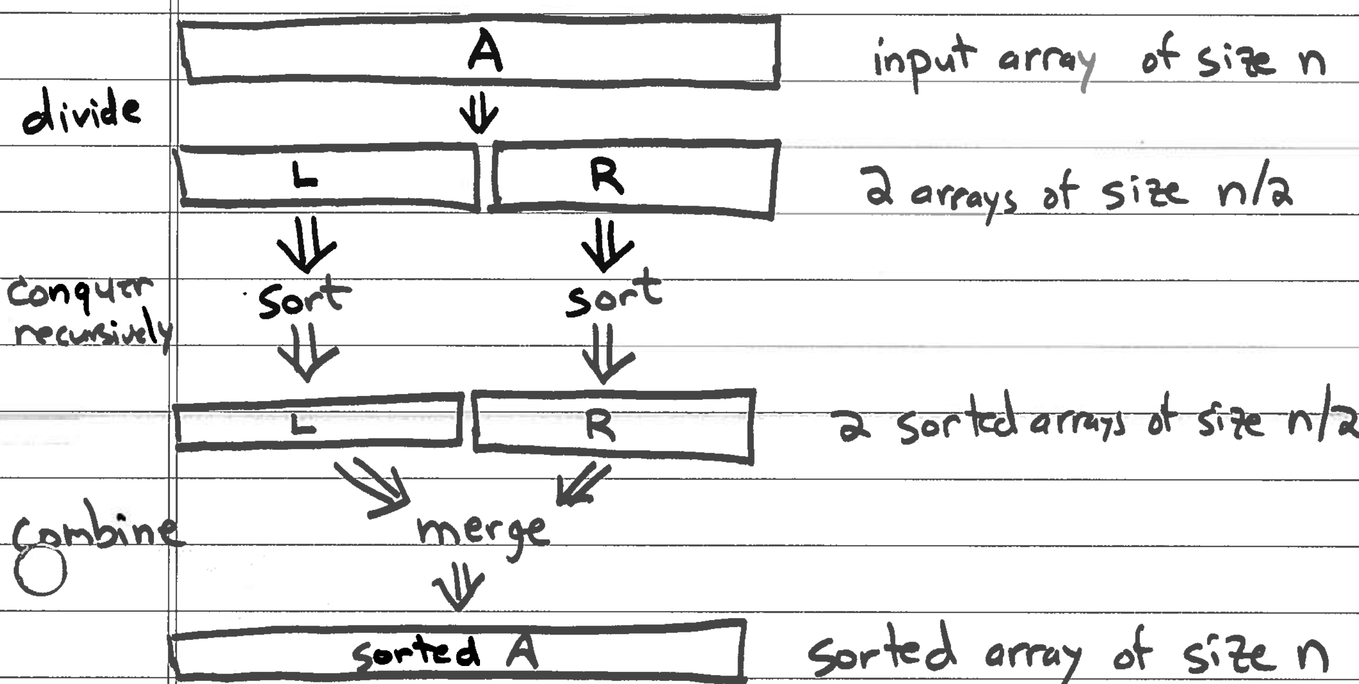
For insertion sort, $T(n) = \Theta(n^2)$
($\approx$ if you double input size, running time goes up $4x$.)

Can we do better? Yes!

6.006
Rivest
L2.4
9/9/08

Divide/Conquer/Combine paradigm

aka "Divide & Conquer"
by example: mergesort



show code (handout): merge_sort
merge ("two finger algorithm")

Ex. merge

5 4 7 3 6 1 9 2

⇓ ⇓

3 4 5 7 1 2 6 9

↑ i ↑ j

(show merge)

1 2 3 4 5 6 7 9

Analysis:

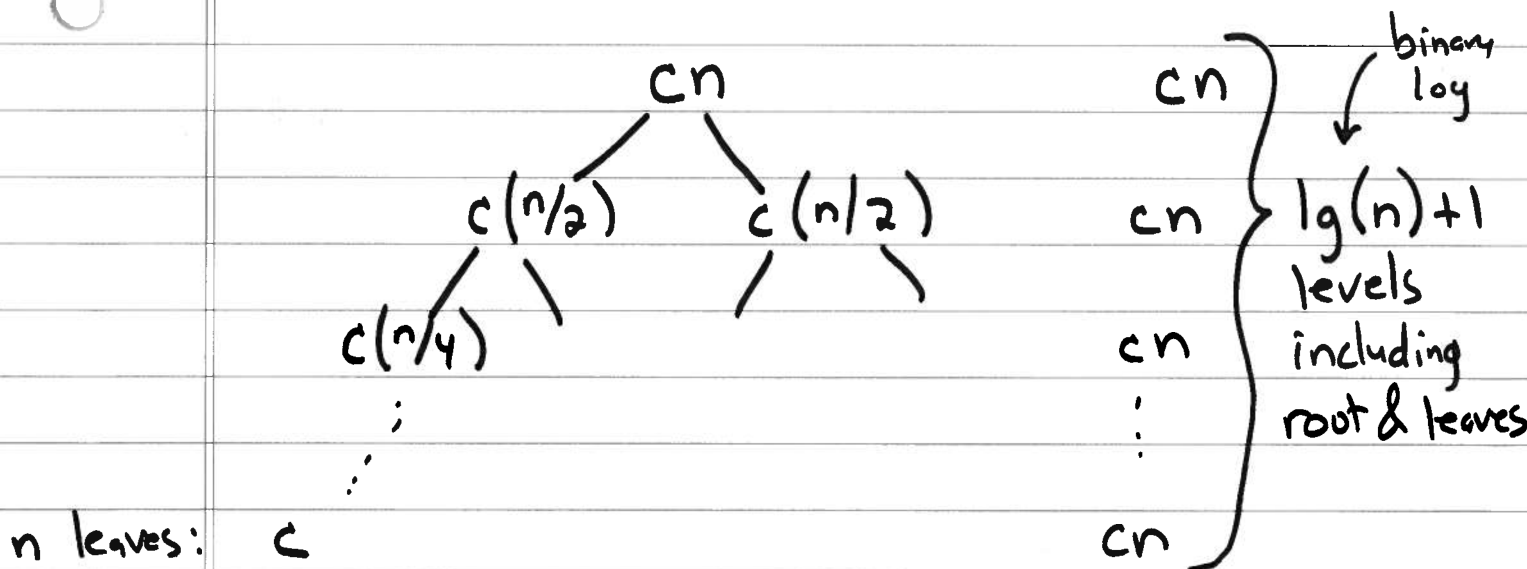
Running time of merge on two inputs of size $n/2$ is $c \cdot n$, for some c.

Let $T(n)$ = running time of mergesort on inputs of size n.

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

$(T(1) = c)$ $\quad = 2T(n/2) + c \cdot n$ (only keep high-order terms)

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



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

$$= \Theta(n \lg n)$$

$\boxed{\begin{array}{c}\underline{\text{Ref:}} \\ \text{CLRS} \\ \text{Chapter 4}\end{array}}$

Experimental Results

Insertion-sort	$\Theta(n^2)$	6.006
merge_sort	$\Theta(n \lg(n))$	Rivest
"sorted" (built-in)	$\Theta(n \lg(n))$?	[redacted] L2.6
		9/9/08

[redacted]

insertion_sort

test_insertion(2**12) ≈ 1 second

insertion sort takes ≈ $66 \cdot n^2$ nanoseconds

... test(test_insertion)...

merge_sort

test_merge(2**17) ≈ 1.5 seconds

merge_sort takes ≈ $701 \cdot n \lg n$ nanoseconds

...test(test_merge)...

Sorted (built-in)

test_sorted(2**20) ≈ 1 second

sorted taks ≈ $55 \cdot n \lg n$ nanoseconds

... test(test_sorted)...

- Not quite linear, as $\lg(n)$ grows slowly, but "almost".

- Small constant for "sorted", since it is written in C. (13x speedup?) but asymptotics same as for mergesort.

6.006
Rivest
L2.7
9/9/08

When is mergesort (in Python)
$701\, n \lg(n)$ nanoseconds
better than insertion_sort in C?
$5 n^2$ nanoseconds $(5 \approx 66/13)$

crossover: $5n^2 > 701\, n \lg n$
at $n \geq 1500$

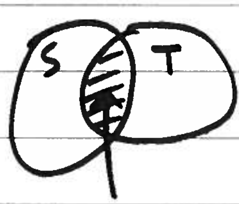
Mergesort wins for $n \geq 1500$

Better algorithm <u>much</u> more valuable than hardware or compiler, even for modest n.

[Note: hybrid approach: use insertion sort if $n \leq 1500$
merge_sort if $n \geq 1500$]

- Python Cost Model – similar experiments on other operations
  - uses timing.py to "fit" formula to data
  - (code might not be so readable...)
  - look at chart...

- <u>Homework:</u> S = set([1,2,3]) <u>set</u> data type
  T = set([1,2,4,9])
  S.intersection(T) = set([1,2])



$S \cap T$

running time may depend on
$|S|$, $|T|$, and $|S \cap T|$

figure it out!

// end of first module