1/14/16

Midterm on Feb 9th: everything covered in lecture
grades posted on myUCLA
homework returned in section

finishing up to ch 2

ex: given a set of integers sort in non decreasing order

6 2 4 9 1 5 7 3

- reduce the problem size so we can use induct
- "divide and conquer" (must be easier than initial problem)

if size of problem is small, sorting is much easier
runtime is not affected by size of partition
i.e. $n/2$ vs $n/3$ vs $n/4$

6 2 4 9 ↑ 1 5 7 3

6 2 ↑ 4 9 ↑ 1 5 ↑ 7 3

6 ↑ 2 ↑ 4 ↑ 9 ↑ 1 ↑ 5 ↑ 7 ↑ 3 this is sorted since they are 8 separate lists

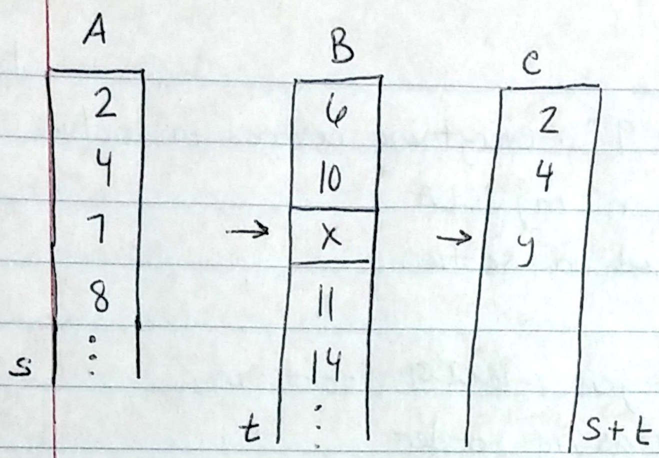
both A and B are in nondecreasing order

A	B	C (sorted list of A and B)
2	6	2
4	10	4
7	11	
⋮	14	
	⋮	
s	t	

$\min(\vec{A}, \vec{B})$
$\rightarrow$ ++

"Merge Algorithm"

if a list finishes, continue the comparison but assume empty list s + t contains infinity



worst case $O(st)$
(very cynical analysis)

how many times do we compare x to something else?
at most s comparisons

using the <u>same</u> algorithm

Can you improve the time analysis? (upperbound)
for any y I spend $O(1)$ time
so since there are $(s+t)$ ys
then $O(s+t)$ is also a time analysis
that is much more optimistic

6 2 4 9 15 7 3

$\boxed{2\ 6}$ $\boxed{4\ 9}$ $\boxed{1\ 5}$ $\boxed{3\ 7}$

$\boxed{2\ 4\ 6\ 9}$ $\boxed{1\ 3\ 5\ 7}$

$\boxed{1\ 2\ 3\ 4\ 5\ 6\ 7\ 9}$

sorted and combined using the Marriage Sort algorithm
(aka Divide and Conquer)
(aka Merge Sort)

<u>Runtime Analysis</u>

$$T(n) = \underbrace{2T\left(\frac{n}{2}\right)}_{\text{dividing/sorting}} + \underbrace{O(n)}_{\text{merging}}$$

$$T(1) = O(1)$$

recursively sub-in $T(n)$

$$T(n) = 2T(n/2) + cn$$
$$= 2[2T(n/4) + cn/2] + cn$$
$$= 2^2 T(n/2^2) + 2cn$$
$$= 2^3 T(n/2^3) + 3cn$$

$$\Rightarrow T(n) = 2^i T(n/2^i) + icn$$

/* always assume logn is base 2 <u>not</u> base 10 */

$$\frac{n}{2^i} \stackrel{?}{=} 1 \qquad i = \log n$$

$$T(n) = 2^{\log n}\, T\left(\frac{n}{2^{\log n}}\right) + \log n \cdot c \cdot n$$
$$= n + cn\log n$$

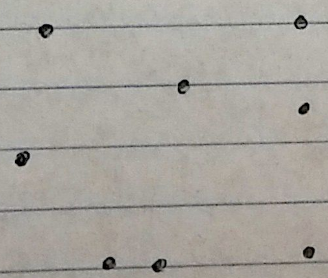
runtime of algorithm: $O(n\log n)$
[merge sort]

can't sort faster than $O(n\log n)$ assuming a general sorting

may be faster if already sorted! (best case)

ex: Closest Pair Problem

assume you have a set of points in the plane (2-dimensional) and find the closest pair



* take one pt and find its distance to all other pts which is $O(1)$ and update the minimum

total runtime: $O(n^2)$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

ex: find minimum value in list

12 4 1 9 11 6 18 3

(n-1) comparisons /*find min in about n */

to find max (n-2) /* find max in n */

$\boxed{12 \quad 4}$ $\boxed{1 \quad 9}$

min: $\boxed{4}$

max: $\boxed{12}$

$$\frac{n}{2} + n = \frac{3n}{2}$$

algorithm: pair up and do comparisons w/in the pair

smaller → candidate for min

larger → candidate for max