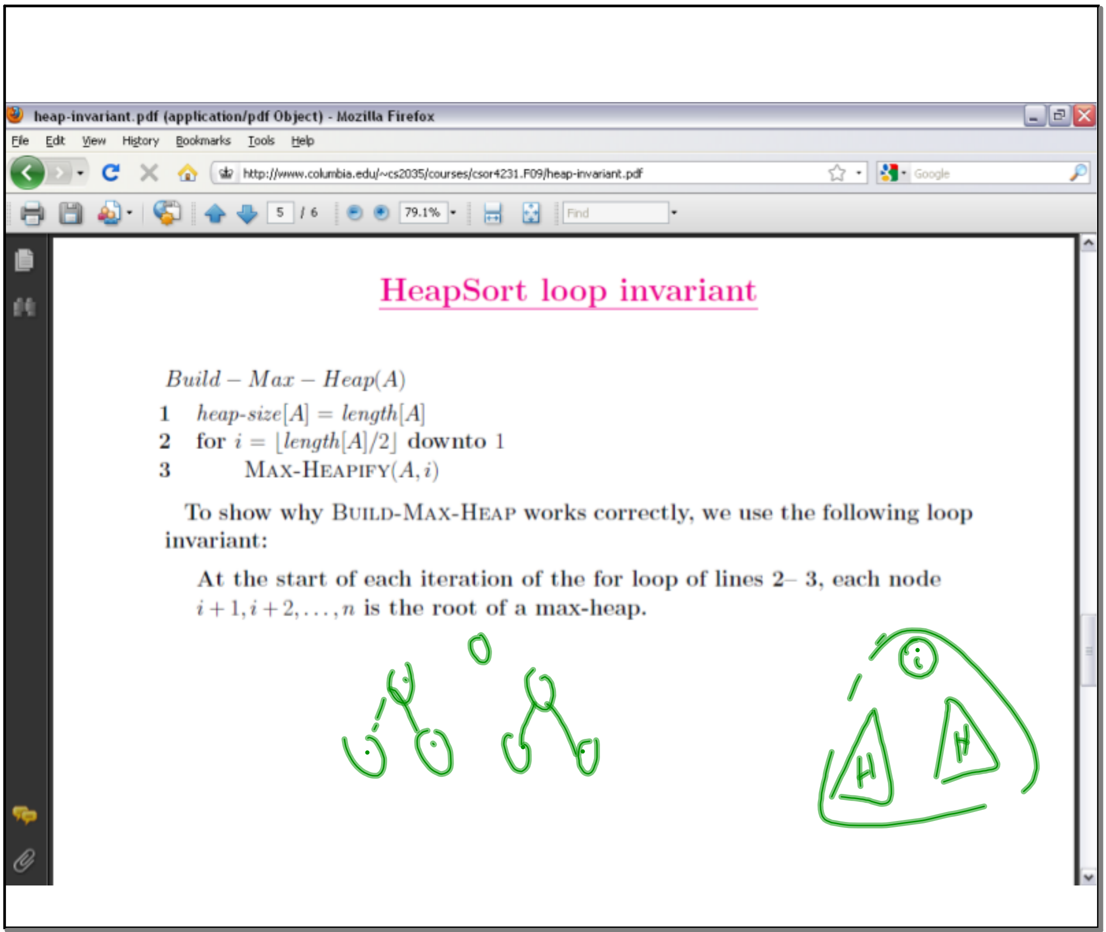


HeapSort loop invariant

Build − *Max* − *Heap*(A)

1 $heap\text{-}size[A] = length[A]$
2 for $i = \lfloor length[A]/2 \rfloor$ downto 1
3 MAX-HEAPIFY(A, i)

To show why BUILD-MAX-HEAP works correctly, we use the following loop invariant:

> At the start of each iteration of the for loop of lines 2– 3, each node $i+1, i+2, \ldots, n$ is the root of a max-heap.

Sep 22-4:08 PM

Sorting

MergeSort
Heapsort $O(n\lg n)$ time

General sorts seem to have running times of $\Omega(n\lg n)$

Nice to say "Any sorting alg. takes $\Omega(n\lg n)$ time"

Equiv. Say an alg is $\Omega(n\lg n)$ means that for infinitely many n, there exists at least one input of size n that takes $\Omega(n\lg n)$ time.

Sep 22-4:16 PM

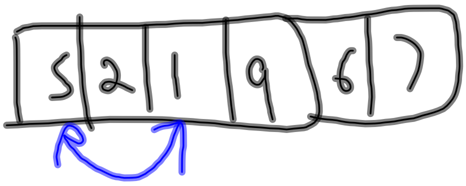
Want to say All algorithms that sort take $\Omega(n \lg n)$ time.

- take all numbers
- square every 3rd
- Look up page A[7] in NY Times of 9/16/2009
- Double it

⋮

Sep 22-4:21 PM

Think about sorts, they do comparisons & swaps.

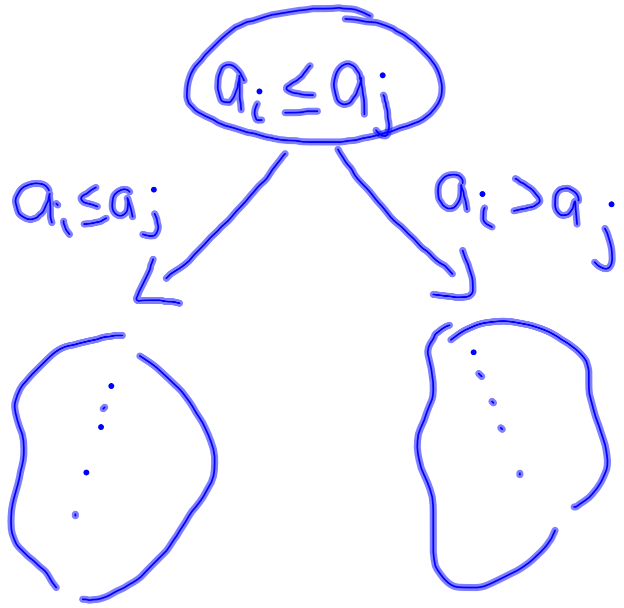


Restrict attention to algs. that only access the data via comparisons

- Convenience: all elements are distinct, ≤

Sep 22-4:24 PM

{ Any permutation of the data } $a_1 \ldots a_n$

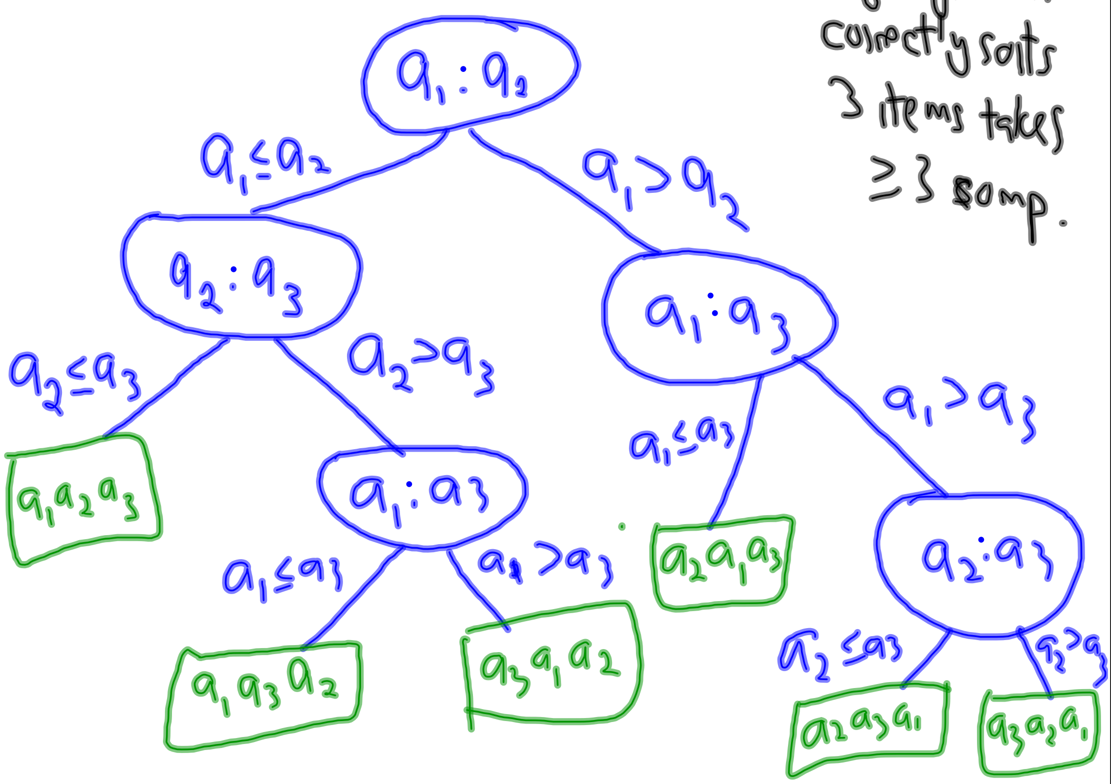


how many comparisons must I make to sort correctly?

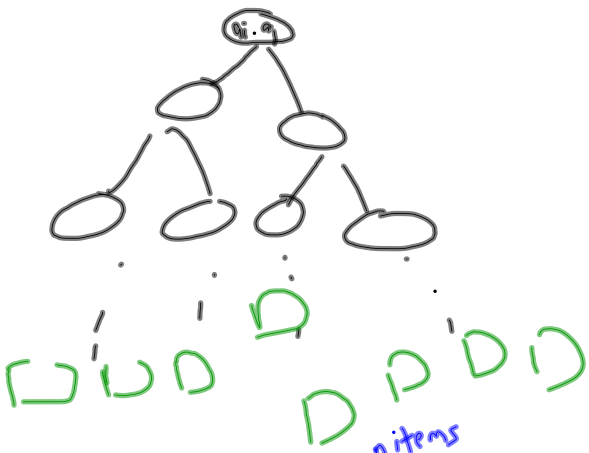
Sep 22-4:29 PM

$a_1 \ a_2 \ a_3$

Any alg. that correctly sorts 3 items takes ≥ 3 comp.

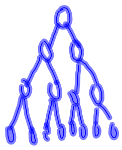


Sep 22-4:33 PM



In order to correctly sort n items, a decision tree must have $\geq n!$ leaves

<u>height</u> of a tree is the length of the longest root leaf path



$$2^h \geq \#\text{leaves} \geq n!$$

$$2^h \geq n!$$

Sep 22-4:39 PM

$$2^h \geq n!$$

$$h \geq \lg(n!)$$

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

Sterling's Approx.

$$n! = \left(\frac{n}{e}\right)^n \sqrt{2\pi n}\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

$$n! \approx \left(\frac{n}{e}\right)^n$$

$$\lg(n!) \approx \lg\left(\frac{n}{e}\right)^n \approx n(\lg n - \lg e) = \Theta(n \lg n)$$

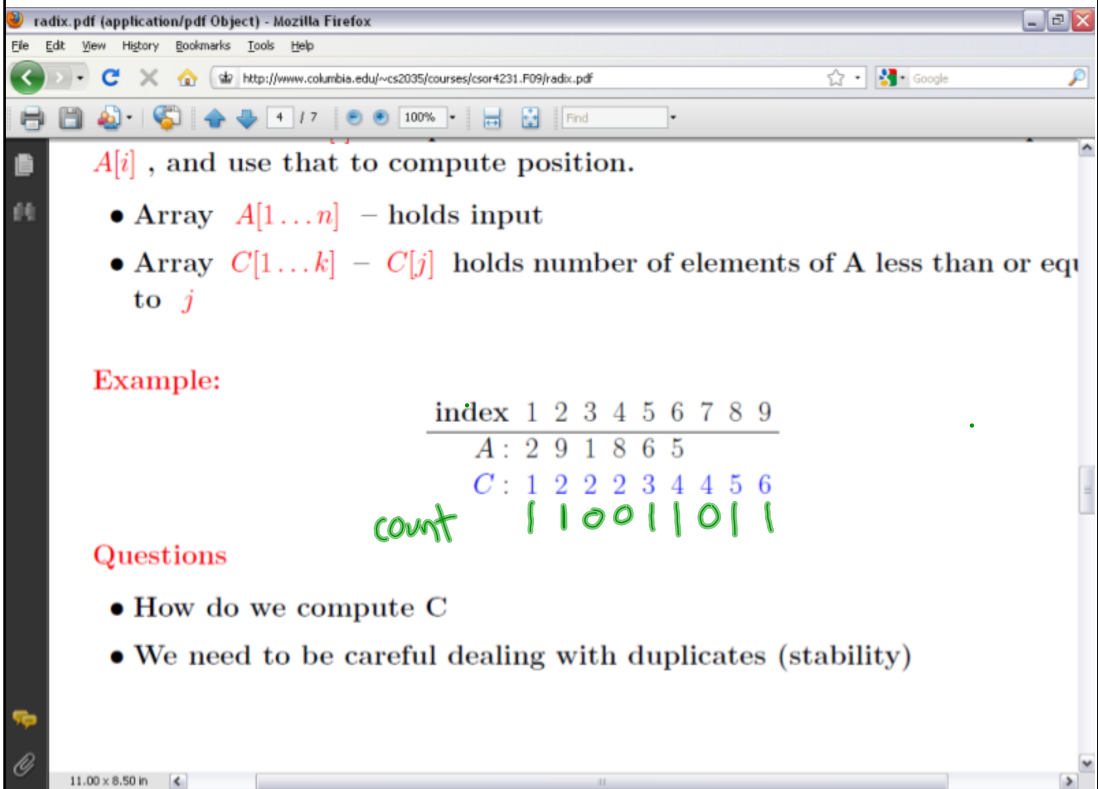
Sep 22-4:44 PM

Any alg. that only access the data via comparisons must take $\Omega(n\lg n)$ time.

Probably implies: Any alg. that sorts any kind of ordered data takes $\Omega(n\lg n)$ time.

Maybe: if we know something about our data, we might be able to sort faster.

Sep 22-4:48 PM



$A[i]$, and use that to compute position.

- Array $A[1\ldots n]$ – holds input
- Array $C[1\ldots k]$ – $C[j]$ holds number of elements of A less than or equ to j

Example:

index	1	2	3	4	5	6	7	8	9
A :	2	9	1	8	6	5			
C :	1	2	2	2	3	4	4	5	6
count	1	1	0	0	1	1	0	1	1

Questions

- How do we compute C
- We need to be careful dealing with duplicates (stability)

Sep 22-5:00 PM

radix.pdf (application/pdf Object) - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://www.columbia.edu/~cs2035/courses/csor4231.F09/radix.pdf

Counting Sort

Counting − *Sort*(A, B, k)

```
for i ← 0 to k                                  O(k)
    do C[i] ← 0
for j ← 1 to length[A]
    do C[A[j]] ← C[A[j]] + 1                    O(n)
▷ C[i] now contains the number of elements equal to i.
for i ← 1 to k
    do C[i] ← C[i] + C[i − 1]                   O(k)
▷ C[i] now contains the number of elements less than or equal to i.
for j ← length[A] downto 1
    do B[C[A[j]]] ← A[j]                        O(n)
       C[A[j]] ← C[A[j]] − 1
```

$O(n+k)$

A: 1 2 3 4 5 6 7 8 9 — 2 4 2 8 3 8 2

C

B: 2 2 2 3 4 8 8

11.00 x 8.50 in

Sep 22-5:08 PM

10^{10} ints from 1 to 9

$O(10^{10} + 9)$ better than $O(10^{10} \lg(10^{10}))$

ints from 1 to 10^{10}

10^3

$O(10^3 + 10^{10})$ vs. $O(10^3 \lg(10^3))$

$O(10^{10})$ mem   mem $O(10^3)$

Sep 22-5:09 PM

Work for data that can be mapped to ints.

e.g. English words $\Leftrightarrow$ base_{27} int

a .. z ␣

cliff

$2 \times 27^4 + 12 \times 27^3 + 9 \times 27^2 + 6 \times 27 + 6$

alphabetize class:

18 $\quad 27^{18} > 10^{18}$

Sep 22-5:14 PM

Radix $\quad 1..b^d \quad O(d(n+b))$

names

100 $\quad 1..27^{18}$ $\quad$ C.S. $27^{18}+100$

R.S. $18(100+27)$

R.S. useful for large ints, words

Sep 22-5:21 PM