notes on program

int [ ] A = new int [1000 1];

* waste A[0]

use A[1]... A[10000]

* never use attribute to get length, keep track with a variable

* o.k. to pass size as a parameter

11:26

BUILD-MAX-HEAP (A)

2/4 = 0

2.0/4.0 = .5

notes

COUNTING-SORT (A, B, k)

for j = 1 to A.length ← end of "this" for loop

Looking @ how deals w/ an array

A	2	5	3	0	2	3	0	3

k=5

	0	1	2	3	4	5
C	II		II	III		I

=>

2	0	2	3	0	1

	1	2	3	4	5	6	7	8
B							3	

A	2	5	3	0	2	3	0	3

j indexes into A

get 3 from

A								3

↑

j

	0	1	2	3	4	5
C	0	2	2	4	7	7

	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	5

IBM used EBCDIC

IBM sorter

~~10 piles based on col #1 { 0 in col #1~~
~~1 in col #1~~
~~:~~
~~:~~
~~8 in col #1~~
~~9 in col #1 }~~
~~→ 10 piles { 0 in col #1, 0 in col #2~~
~~0 in col #1, 1 in col #2~~
~~0 in col #1, 9 in col #2 }~~

Too Many Subcolumns to keep track of!

steps

① * sort on least significant column first

② (take all piles & restack <u>in</u> order of what's in sorted column

③ sort on next least significant pile column

④ repeat from step 2 until sorted all columns

Time complexity of above steps is $\Theta(n)$ assuming fixed, finite <u>radix</u> (base) that the #s are represented in

Assuming you use a STABLE sort for each digit

STABLE SORT - leaves duplicates in the sorted array in the same order the appeared in the input array (only matters when extra info stored w/ each sorting key)

329			720		720		329
457	✓		355		329		355
657	✓		436		436		436
839		→	457	→	839	→	457
436	✓		657		355		657
720	✓		329		457		720
355	✓		839		657		839

SORTED! YAY!

So Stable Sort's time complexity = $\Theta(n)$

Let's say we have integers 0 to k, but don't want to do radix sort b/c there is sparsness between #s

So we

Divide input into sub-ranges (call them "buckets")

* Assume input numbers to be sorted are <u>uniformly distributed</u> over range Ø...k (range 0...1 for pseudo-code EX's)

→ Any value in the input array is equally likely to fall into any bucket

After bucket sort, Ch. 22