notes on program

int [ ] A = new int [10001];

* 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

C | 0 | 1 | 2 | 3 | 4 | 5 |
  | 11 | | 11 | 111 | | 1 |

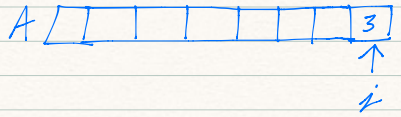
=> | 2 | 0 | 2 | 3 | 0 | 1 |

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

A | 2 | 5 | 3 | 0 | 2 | 3 | 0 | 3 |

j indexes into A

get 3 from

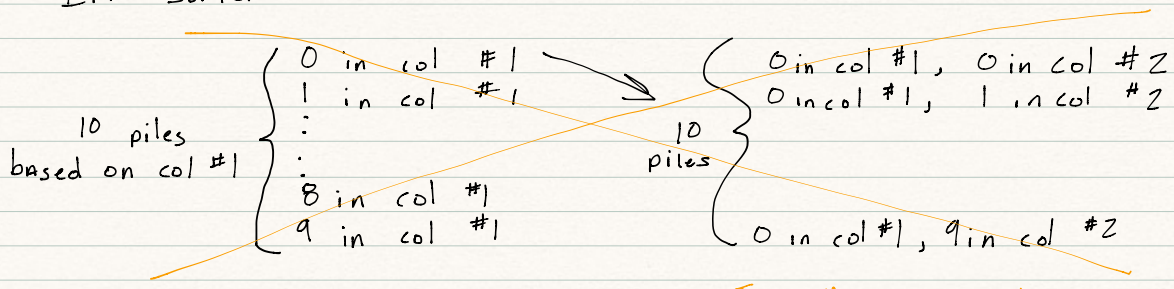


C | 0 | 1 | 2 | 3 | 4 | 5 |
  | 0 | 2 | 2 | 4 | 7 | 7 |

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

IBM used EBCDIC

IBM sorter



Too Many Subcolumns to keep track of!

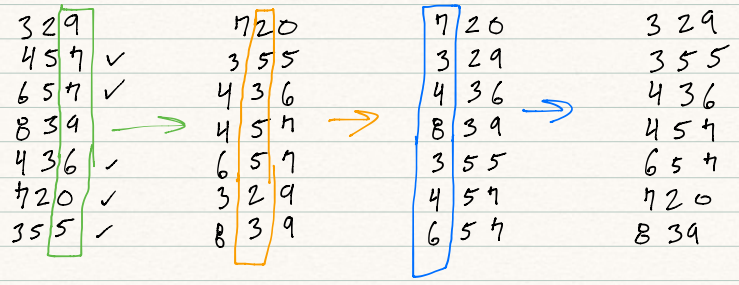
steps

1. * sort on least significant column first
2. (take all piles & restack in 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 radix (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)



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 uniformly distributed over range 0...k (range 0...1 for psendo-code EX's)

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

After bucket sort, Ch. 22