CSE 326: Data Structures
Sorting in (kind of) linear time

Luke McDowell
Summer Quarter 2003

BinSort (aka BucketSort)

- If all keys are between 1 and K
- Have array of size K
- Put keys into correct bin (cell) of array

Example K=5. Values = (5,1,3,4,3,2,1,1,5,4,5)

Bins in array	
key = 1	
key = 2	
key = 3	
key = 4	
key = 5	

Running time?

BinSort Running Time:

- Case 1: K is a constant
  - BinSort is linear time
- Case 2: K is variable
  - Not simply linear time
- Case 3: K is large (e.g. 2^{32})
  - ???

Digression: Stable Sorting

- Stable Sorting algorithm.
  - Items in input with the same key end up in the same order as when they began.
  - Important if keys have associated values
- Are these stable?
  - RadixSort?
  - MergeSort?
  - QuickSort?

RadixSort

- Radix = "The base of a number system" (Webster's dictionary)
  - We'll use 10 for convenience, but could be anything
- Random Trivia?
- Idea: BinSort on each digit, bottom up.

RadixSort – magic! It works.

- Input:126, 328, 636, 341, 416, 131, 328

BinSort on lowest digit:

0	1	2	3	4	5	6	7	8	9

BinSort on next-higher digit:

0	1	2	3	4	5	6	7	8	9

BinSort on highest digit:

0	1	2	3	4	5	6	7	8	9

Not magic. It provably works.

- Keys
  - n-digit numbers
  - base K
- Claim: after i^{th} BinSort, least significant i digits are sorted.
  - e.g. K=10, i=3, keys are 1776 and 8234. 8234 comes before 1776 for last 3 digits.

Induction to the rescue…

- Base case
  - i=0. 0 digits are sorted
- Induction step
  - assume for i, prove for i+1.
  - consider two numbers: X, Y. Say X_i is i^{th} digit of X (from the right)
    - $X_{i+1} < Y_{i+1}$ then $i+1^{th}$ BinSort will put them in order
    - $X_{i+1} > Y_{i+1}$, same thing
    - $X_{i+1} = Y_{i+1}$, order depends on last i digits. Induction hypothesis says already sorted for these digits. (Careful about ensuring that your BinSort preserves order aka "stable"…)

Time to play at home…

- RadixSort the following values using K=10:
  95, 3, 927, 187, 604, 823, 805, 422, 159, 98, 123, 3, 987, 125.

  (space on next slide)
- Given arbitrary numbers A_1, A_2, …A_n, and a base K, what is the overall running time of radix sort?
- How should you choose the value of K?

(extra space)

Running time of Radixsort

- How many passes?
- How much work per pass?
- Total time?
- Conclusion?
- In practice
  - RadixSort only good for large number of items, relatively small keys
  - Hard on the cache, vs. MergeSort/QuickSort

What data types can you RadixSort?

- Any type T that can be BinSorted
- Any type T that can be broken into parts A and B,
  - You can reconstruct T from A and B
  - A can be RadixSorted
  - B can be RadixSorted
  - A is always more significant than B, in ordering

Example:

- 1-digit numbers can be BinSorted
- 2 to 5-digit numbers can be BinSorted without using too much memory
- 6-digit numbers, broken up into A=first 3 digits, B=last 3 digits.
  - A and B can reconstruct original 6-digits
  - A and B each RadixSortable as above
  - A more significant than B

RadixSorting Strings

- 1 Character can be BinSorted
- Break strings into characters
- Need to know length of biggest string (or calculate this on the fly).

RadixSorting Strings example

	5th pass	4th pass	3rd pass	2nd pass	1st pass
String 1	z	i	p	p	y
String 2	z	a	p		
String 3	a	n	t	s	
String 4	f	l	a	p	s

NULLs are just like fake characters

RadixSorting Strings running time

- N is number of strings
- L is length of longest string
- Total Running time:

- L ~ 20. Is this better than Quicksort?