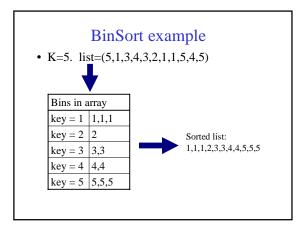


BinSort (a.k.a. BucketSort)

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



BinSort Running Time:

- K is a constant
 BinSort is linear time
- K is variable
 Not simply linear time
- K is large (e.g. 2³²) – Impractical

BinSort is "stable"

• 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
- Critical for RadixSort

RadixSort

- Radix = "The base of a number system" (Webster's dictionary)
- History: used in 1890 U.S. census by Hollerith
- Idea: BinSort on each digit, bottom up.

RadixSort – magic! It works.

- Input list: 126, 328, 636, 341, 416, 131, 328
- BinSort on lower digit: 341, 131, 126, 636, 416, 328, 328
- BinSort result on next-higher digit: 416, 126, 328, 328, 131, 636, 341
- BinSort that result on highest digit: 126, 131, 328, 328, 341, 416, 636

Not magic. It provably works.

- Keys
 - n-digit numbers
 - base B
- Claim: after ith BinSort, least significant i digits are sorted.
 - e.g. B=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 (that wasn't hard!)

Induction is rescuing us...

· Induction step

- assume for i, prove for i+1.
- consider two numbers: X, Y. Say X_i is ith digit of X (from the right)
 - $X_{i+1} < Y_{i+1}$ then i+1th 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"...)

Running time of Radixsort

- · How many passes?
- How much work per pass?
- Total time?
- Conclusion
 - Not truly linear if K is large.
- In practice
 - RadixSort only good for large number of items, relatively small keys
 - Hard on the cache, vs. MergeSort/QuickSort
 - Hard on the cache, vs. Mergeson/Quickson

What data types can you RadixSort?

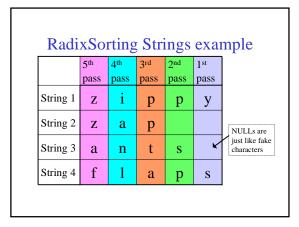
- Any type T that can be BinSorted
- Any type T that can be broken into parts A and B,
 - You can reconstruct T from A and B
 - A can be RadixSorted
 - B can be RadixSorted
 - A is always more significant than B, in ordering

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 running time

- N is number of strings
- L is length of longest string
- RadixSort takes O(N*L)

