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)

<table>
<thead>
<tr>
<th>Bins in array</th>
<th>Running time?</th>
</tr>
</thead>
<tbody>
<tr>
<td>key = 1</td>
<td></td>
</tr>
<tr>
<td>key = 2</td>
<td></td>
</tr>
<tr>
<td>key = 3</td>
<td></td>
</tr>
<tr>
<td>key = 4</td>
<td></td>
</tr>
<tr>
<td>key = 5</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>RadixSort on lowest digit:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RadixSort on next-higher digit:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RadixSort on highest digit:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>
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)
    - $X_i < Y_i$, then $i$th BinSort will put them in order
    - $X_i > Y_i$, same thing
    - $X_i = Y_i$, 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 $K$?

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

<table>
<thead>
<tr>
<th></th>
<th>5th pass</th>
<th>4th pass</th>
<th>3rd pass</th>
<th>2nd pass</th>
<th>1st pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>String 1</td>
<td>z</td>
<td>i</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
<tr>
<td>String 2</td>
<td>z</td>
<td>a</td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>String 3</td>
<td>a</td>
<td>n</td>
<td>t</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>String 4</td>
<td>f</td>
<td>l</td>
<td>a</td>
<td>p</td>
<td>s</td>
</tr>
</tbody>
</table>

RadixSorting Strings running time

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

  - L \sim 20. Is this better than Quicksort?