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 i^{th} 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 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”…)

Paleontology fact

- Early humans had to survive without induction.

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

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
- RadixSort takes O(N*L)

RadixSorting IEEE floats/doubles

- You can RadixSort real numbers, in most representations
- We do IEEE floats/doubles, which are used in C/C++.
- Some people say you can't RadixSort reals. In practice (like IEEE reals) you can.

Anatomy of a real number

Sign (positive or negative)

$-1.3892 * 10^{24}$

$+1.507 * 10^{-17}$

Exponent

Significand (a.k.a. mantissa)

IEEE FLOATS IN BINARY*

$$-1.0110100111 * 2^{1011}$$
$$+1.101101001 * 2^{-1}$$

- Sign: 1 bit
- Significand: always 1.*fraction*. *fraction* uses 23 bits
- Biased exponent: 8 bits.
  - Bias: represent –127 to +127 by adding 127 (so range is 0-254)

* okay, simplified to focus on the essential ideas.

Observations

- significand always starts with 1
  → only one way to represent any number
- Exponent always more significant than significand
- Sign is most significant, but in a weird way

Pseudocode

```
procedure RadixSortReals (Array[1..N])

RadixSort Significands in Array as unsigned ints
RadixSort biased exponents in Array as u-ints

Sweep thru Array,
      put negative #'s separate from positive #'s.
Flip order of negative #'s, & put them before
      the positive #'s.

Done.
```