Algorithm and Data Structure Analysis (ADSA)

Lecture 6: Linear-Time Sorting Algorithms (Radix sort)

David Luebke

What is a stable sort

- A "stable" sorting algorithm keeps the items with the same sorting key in order.

- Let’s look at an example

Why don't we always use counting sort?

- Because it depends on range *k* of elements
- *Could we use counting sort to sort 32 bit integers? Why or why not?*
- Answer: no, *k* too large (2^{32} = 4,294,967,296)

Radix Sort

- *How did IBM get rich originally?*
- Answer: punched card readers for census tabulation in early 1900's.
  - In particular, a *card sorter* that could sort cards into different bins
    - Each column can be punched in 12 places
    - Decimal digits use 10 places
  - Problem: only one column can be sorted on at a time

Radix Sort

Lets work through an example

- 21, 12, 22,23, 13, 31, 12

Radix Sort

- Intuitively, you might sort on the most significant digit, then the second msd, etc.
  - Problem: ????
- Key idea: sort the *least* significant digit first

```
RadixSort(A, d)
    for i=1 to d
        StableSort(A) on digit i
```

Radix Sort

Can you prove it will work?

Sketch of an inductive argument (induction on the number of passes):

- Assume lower-order digits {j: j<i}are sorted
- Show that sorting next digit i leaves array correctly sorted
  - If two digits at position i are different, ordering numbers by that digit is correct (lower--order digits irrelevant)
  - If they are the same, numbers are already sorted on the lower--order digits. Since we use a stable sort, the numbers stay in the right order

Radix Sort

- *What sort will we use to sort on digits?*
- Counting sort is obvious choice:
  - Sort n numbers on digits that range from 1..k
  - Time: O(n + k)
- Each pass over n numbers with d digits takes time O(n+k), so total time O(dn+dk)
  - When d is constant and k=O(n), takes O(n) time
- *How many bits in a computer word?*

Radix Sort

- Problem: sort 1 million 64-bit numbers
  - Treat as four-digit radix 2^{16} numbers
  - Can sort in just four passes with radix sort!
- Compares well with typical O(n lg n) comparison sort
  - Requires approx lg n = 20 operations per number being sorted

Radix Sort

- In general, radix sort based on counting sort is
  - Fast
  - Asymptotically fast (i.e., O(*n*))
  - Simple to code
  - A good choice

- *So why would we ever use anything but radix sort?*