

CSE373: Data Structures and Algorithms

## Bucket Sort and Radix Sort

Steve Tanimoto  
Winter 2016

This lecture material represents the work of multiple instructors at the University of Washington. Thank you to all who have contributed!

### Sorting: The Big Picture

Surprising amount of neat stuff to say about sorting:

<b>Simple algorithms:</b> $O(n^2)$	<b>Fancier algorithms:</b> $O(n \log n)$	<b>Comparison lower bound:</b> $\Omega(n \log n)$	<b>Specialized algorithms:</b> $O(n)$	<b>Handling huge data sets</b>
Insertion sort Selection sort Shell sort ...	Heap sort Merge sort Quick sort ...		Bucket sort Radix sort	External sorting

Winter 2016
CSE 373: Data Structures & Algorithms
2

### Radix sort

- Origins go back to the 1890 U.S. census
- Radix = "the base of a number system"
  - Examples will use 10 because we are used to that
  - In implementations use larger numbers
    - For example, for ASCII strings, might use 128
- Idea:
  - Bucket sort on one digit at a time
    - Number of buckets = radix
    - Starting with *least* significant digit
    - Keeping sort *stable*
  - Do one pass per digit
  - Invariant: After  $k$  passes (digits), the last  $k$  digits are sorted

Winter 2016
CSE 373: Data Structures & Algorithms
3

### Example

Radix = 10

0	1	2	3	4	5	6	7	8	9
	721		3				537	478	9
			143				67	38	

Input: 478 537 9 721 3 38 143 67

First pass: bucket sort by ones digit

Order now: 721 3 143 537 67 478 38 9

Winter 2016
CSE 373: Data Structures & Algorithms
4

### Example

Radix = 10

0	1	2	3	4	5	6	7	8	9
	721		3				537	478	9
			143				67	38	

Order was: 721 3 143 537 67 478 38 9

Second pass: *stable* bucket sort by tens digit

Order now: 3 9 721 537 67 143 478 38

Winter 2016
CSE 373: Data Structures & Algorithms
5

### Example

Radix = 10

0	1	2	3	4	5	6	7	8	9
3		721	537	143		67	478		
9			38						

Order was: 3 9 721 537 38 67 143 478

Third pass: *stable* bucket sort by 100s digit

Order now: 3 9 38 67 143 478 537 721

Winter 2016
CSE 373: Data Structures & Algorithms
6

### Analysis

Input size:  $n$

Number of buckets = Radix:  $B$

Number of passes = "Digits":  $P$

Work per pass is 1 bucket sort:  $O(B+n)$

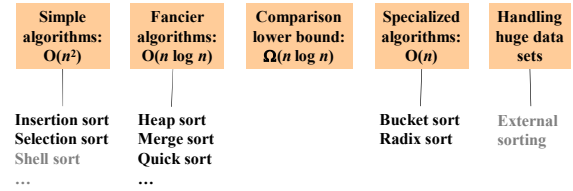
Total work is  $O(P(B+n))$

Compared to comparison sorts, sometimes a win, but often not

- Example: Strings of English letters up to length 15
  - Run-time proportional to:  $15 \cdot (52 + n)$
  - This is less than  $n \log n$  only if  $n > 33,000$
  - Of course, cross-over point depends on constant factors of the implementations
    - And radix sort can have poor locality properties

### Sorting: The Big Picture

Surprising amount of neat stuff to say about sorting:



### Last Slide on Sorting

- Simple  $O(n^2)$  sorts can be fastest for small  $n$ 
  - Selection sort, Insertion sort (latter linear for mostly-sorted)
  - Good for "below a cut-off" to help divide-and-conquer sorts
- $O(n \log n)$  sorts
  - Heap sort, in-place but not stable nor parallelizable
  - Merge sort, not in place but stable and works as external sort
  - Quick sort, in place but not stable and  $O(n^2)$  in worst-case
    - Often fastest, but depends on costs of comparisons/copies
- $\Omega(n \log n)$  is worst-case and average lower-bound for sorting by comparisons
- Non-comparison sorts
  - Bucket sort good for small number of possible key values
  - Radix sort uses fewer buckets and more phases
- Best way to sort? It depends!

### Done with sorting! (phew..)

- Moving on....
- There are many many algorithm techniques in the world
  - We've learned a few
- What are a few other "classic" algorithm techniques you should at least have heard of?
  - And what are the main ideas behind how they work?