Bucket and Radix Sorting

Prof. Dianne Foreback

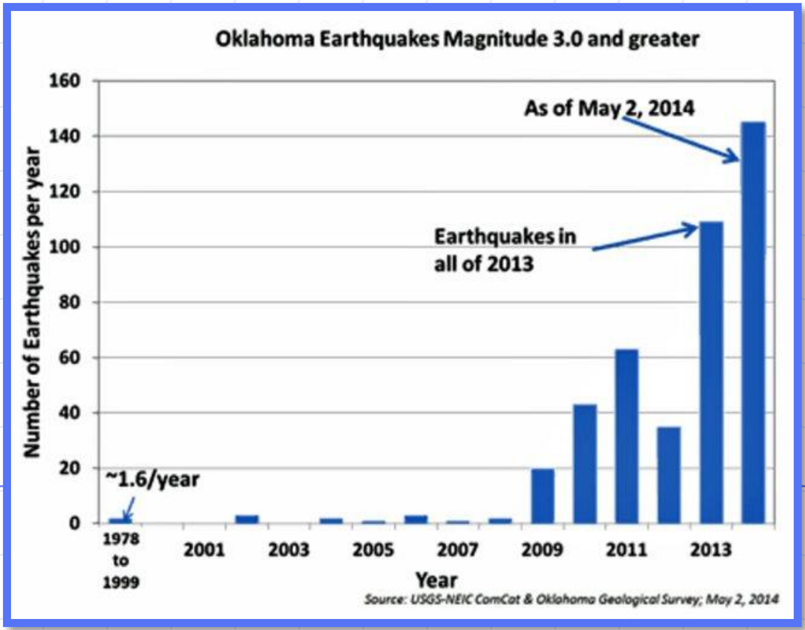
Slides from *Algorithm Design & Applications* by Michael T. Goodrich and Roberto Tamassia, © 2015 John Wiley & Sons, Inc.Goodrich and Tamassia, ISBN: 978-1118335918.

Reading Material

- *Algorithm Design & Applications* by Michael T. Goodrich and Roberto Tamassia
  - Chapter 9 Section 9.1

Presentation for use with the textbook, Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015

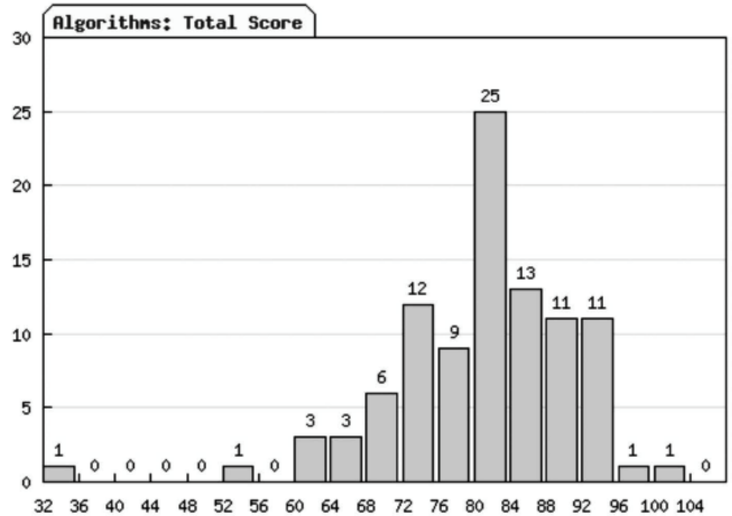
Bucket-Sort and Radix-Sort



USGS NEIC. Public domain government image.

Application: Constructing Histograms

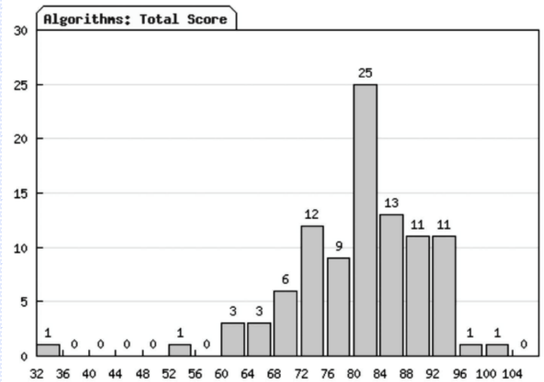
- One common computation in data visualization and analysis is computing a **histogram**.
- For example, n students might be assigned integer scores in some range, such as 0 to 100, and are then placed into ranges or “buckets” based on these scores.



A histogram of scores from a recent Algorithms course taught by one of the authors (with extra credit included).

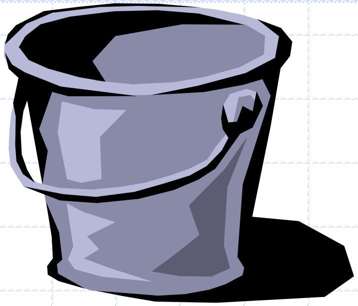
Application: An Algorithm for Constructing Histograms

- When we think about the algorithmic issues in constructing a histogram of n scores, it is easy to see that this is a type of sorting problem.
- But it is not the most general kind of sorting problem, since the keys being used to sort are simply integers in a given range.
- So a natural question to ask is whether we can sort these values faster than with a general comparison-based sorting algorithm.
- The answer is "yes." In fact, we can sort them in O(n) time.



A histogram of scores from a recent Algorithms course taught by one of the authors (with extra credit included).

Bucket-Sort



- Let be S be a sequence of n (key, element) items with keys in the range $[0, N-1]$
- Bucket-sort uses the keys as indices into an auxiliary array B of sequences (buckets)

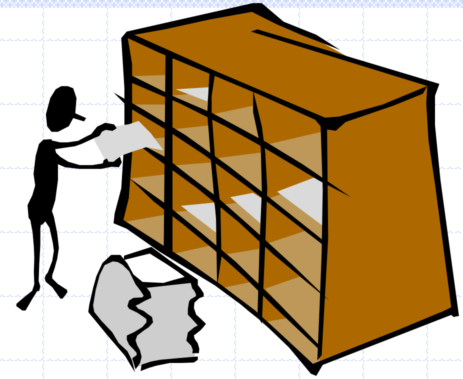
  Phase 1: Empty sequence S by moving each entry (k, o) into its bucket $B[k]$

  Phase 2: For $i = 0, \ldots, N-1$, move the entries of bucket $B[i]$ to the end of sequence S
- Analysis:
  - Phase 1 takes $O(n)$ time
  - Phase 2 takes $O(n + N)$ time

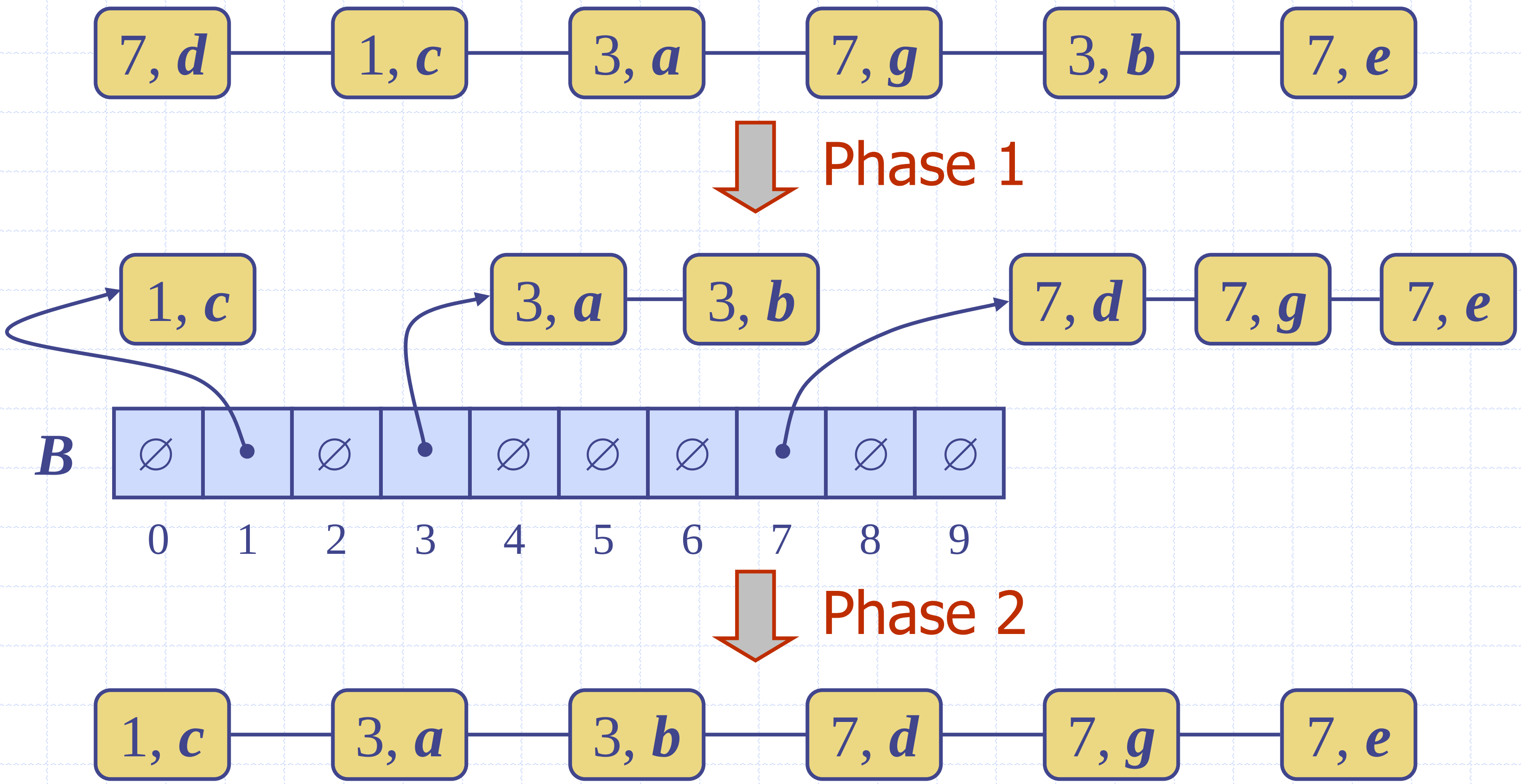
  Bucket-sort takes $O(n + N)$ time

Algorithm bucketSort(S):
Input: Sequence S of entries with integer keys in the range [0, N − 1]
Output: Sequence S sorted in nondecreasing order of the keys
let B be an array of N sequences, each of which is initially empty
for each entry e in S **do**
  k = the key of e
  remove e from S
  insert e at the end of bucket B[k]
for i = 0 to N−1 **do**
  for each entry e in B[i] **do**
    remove e from B[i]
    insert e at the end of S

Example



- Key range [0, 9]



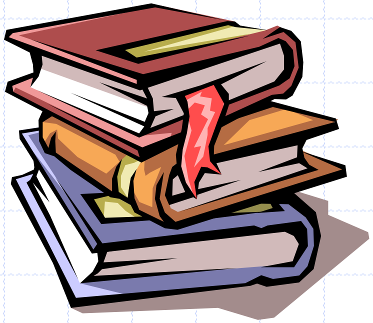
Properties and Extensions

- Key-type Property
  - The keys are used as indices into an array and cannot be arbitrary objects
  - No external comparator
- Stable Sort Property
  - The relative order of any two items with the same key is preserved after the execution of the algorithm

Extensions

- Integer keys in the range $[a, b]$
  - Put entry (k, o) into bucket $B[k - a]$
- String keys from a set D of possible strings, where D has constant size (e.g., names of the 50 U.S. states)
  - Sort D and compute the rank $r(k)$ of each string k of D in the sorted sequence
  - Put entry (k, o) into bucket $B[r(k)]$

Lexicographic Order



- A d-tuple is a sequence of d keys $(k_1, k_2, \ldots, k_d)$, where key k_i is said to be the i-th dimension of the tuple
- Example:
  - The Cartesian coordinates of a point in space are a 3-tuple
- The lexicographic order of two d-tuples is recursively defined as follows

$$(x_1, x_2, \ldots, x_d) < (y_1, y_2, \ldots, y_d)$$
$$\Leftrightarrow$$
$$x_1 < y_1 \vee x_1 = y_1 \wedge (x_2, \ldots, x_d) < (y_2, \ldots, y_d)$$

I.e., the tuples are compared by the first dimension, then by the second dimension, etc.

Lexicographic-Sort

- Let C_i be the comparator that compares two tuples by their i-th dimension
- Let $stableSort(S, C)$ be a stable sorting algorithm that uses comparator C
- Lexicographic-sort sorts a sequence of d-tuples in lexicographic order by executing d times algorithm $stableSort$, one per dimension
- Lexicographic-sort runs in $O(dT(n))$ time, where $T(n)$ is the running time of $stableSort$

Algorithm *lexicographicSort*(S)
  Input sequence S of d-tuples
  Output sequence S sorted in lexicographic order

  for $i \leftarrow d$ **downto** 1
    $stableSort(S, C_i)$

Example:

(7,4,6) (5,1,5) (2,4,6) (2, 1, 4) (3, 2, 4)

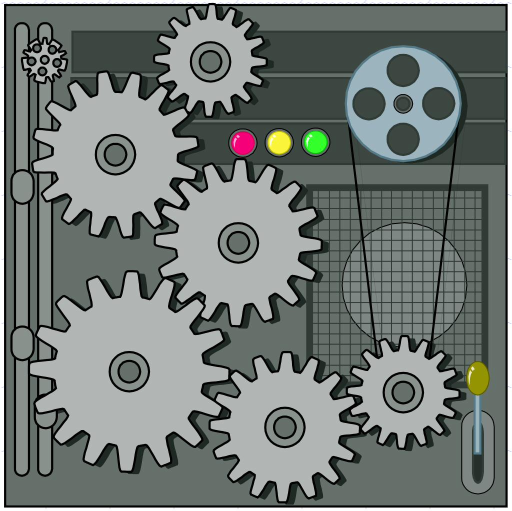
(2, 1, 4) (3, 2, 4) (5,1,5) (7,4,6) (2,4,6)

(2, 1, 4) (5,1,5) (3, 2, 4) (7,4,6) (2,4,6)

(2, 1, 4) (2,4,6) (3, 2, 4) (5,1,5) (7,4,6)

Radix-Sort

- Radix-sort is a specialization of lexicographic-sort that uses bucket-sort as the stable sorting algorithm in each dimension.
- Radix-sort is applicable to tuples where the keys in each dimension i are integers in the range $[0, N-1]$
- Radix-sort runs in time $O(d(n+N))$
- If d is constant and N is $O(n)$, then this is $O(n)$.



Algorithm *radixSort*(S, N)

Input sequence S of d-tuples such that $(0, \ldots, 0) \le (x_1, \ldots, x_d)$ and $(x_1, \ldots, x_d) \le (N-1, \ldots, N-1)$ for each tuple $(x_1, \ldots, x_d)$ in S

Output sequence S sorted in lexicographic order

for $i \leftarrow d$ **downto** 1

bucketSort(S, N)