# CSIS 3103

Ch 8: Non-comparison Sorts

## Comparison Based Sorts

- All previous sorting techniques are comparison based
- The fastest possible comparison sort is $O(n \log n)$
- Are there any faster sorting algorithms?

## Bucket Sort

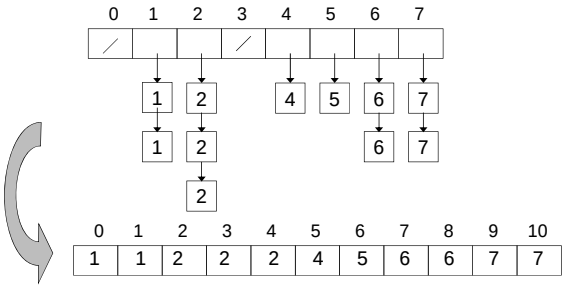
- A non-comparison sort
- Bucket sort works well when
  - Keys are distributed in a range, $0..m\text{-}1$
  - and this range is small compared to the number of items to be sorted (duplicates are allowed)

## Bucket Sort Example

- Suppose all keys are in: **`0..7`**
- Create an array or `ArrayList` with $m = 8$ *buckets*, making each bucket a Queue
- Insert all input values into the appropriate bucket
- Concatenate or copy the queues in order

## Bucket Sort Example

Inputs: 5, 2, 7, 6, 1, 2, 4, 6, 7, 2, 1




## Bucket Sort

Run time is $O(n + m)$ which is $O(n)$ when $m$ is relatively small

Variation (*Counting Sort*):

- If the keys are just raw numbers, just store a count in the buckets

## Radix Sort

Radix sort considers the structure of the keys

- Suppose we want to sort 1000 items in the range from 0 to 99,999,999
- Bucket sort would spend too much time initializing and concatenating empty queues

Sorting punch cards: http://en.wikipedia.org/wiki/File:Punch_card_sorter.JPG

## Radix Sort

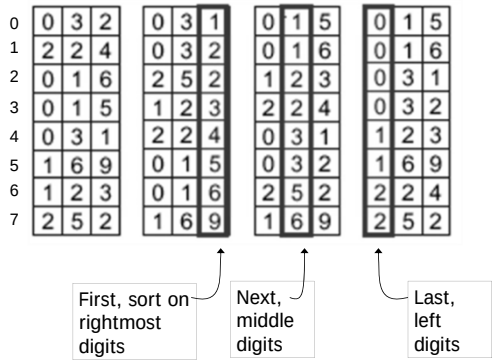
Given input of $n$ numbers having $d$-digits:

for $k$ in $0..d$-1
  sort the array in a *stable* way,
    looking only at digit $k$

(*stable* means equal items are kept in the same order relative to each other as they were before sorting)

- What stable sort should we use?
  - Bucket Sort! It's $O(n)$
- Thus, total running time is $O(dn)$
- And $d$ is a constant, so Radix Sort is $O(n)$

## Radix Sort With 3-digit Integers




## Radix Sort… Made Even Better?

- We can actually do better than sorting on one decimal digit at a time
- It would likely be faster if we sort on two digits at a time (using a radix of 100) or three (using a radix of 1000)
- But on computers, it's more natural to choose a power-of-two radix like 256
  - Base-256 digits are easier to extract from a key, because eight bits can quickly be pulled out of an integer

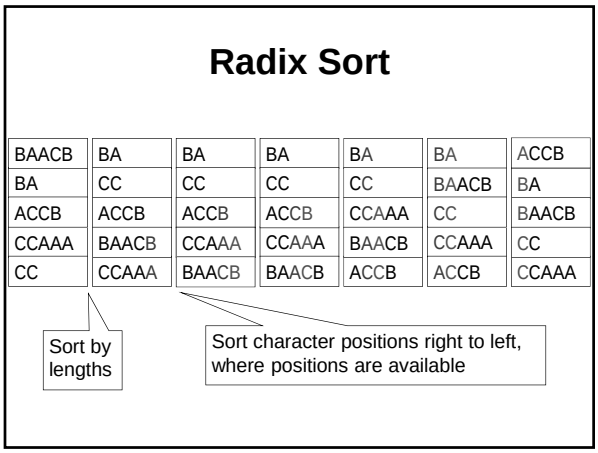
## Radix Sort

- Radix sort is not limited to just integer keys
- Almost any data that can be compared bitwise can be used
  - IEEE standard for floating-point numbers is designed to work with radix sort

## Radix Sort For Strings

Strings of different lengths can be sorted in time proportional to the total length of the strings

- Phase 1: Sort the strings by their length
- Phase 2: Sort the strings character by character (or several characters at a time), starting with the last character of the longest string and working backward to the first character of every string
- We don't sort every string during every pass of the second phase - only if it has a character in the appropriate place


Radix Sort
BAACB BA BA BA BA BA ACCB
BA CC CC CC CC BAACB BA
ACCB ACCB ACCB ACCB CCAAA CC BAACB
CCAAA BAACB CCAAA CCAAA BAACB CCAAA CC
CC CCAAA BAACB BAACB ACCB ACCB CCAAA
Sort by lengths
Sort character positions right to left, where positions are available