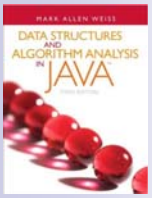
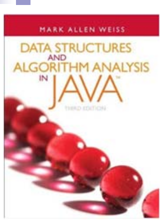



FLORIDA INTERNATIONAL UNIVERSITY

# Sorting (V)

Dr. Antonio L. Bajuelos
FIU School of Computing & Information Sciences






## COP-3530 - Data Structures

**Module #6: Sorting (part V)**

**Outline:**

- The lower bound of sorting
- The Master Theorem
- Sorting in Linear Time
  - Bucket-Sort
  - Radix-Sort
  - Examples
  - Complexity analysis

2

## A General Lower Bound for Sorting

- **How fast can we Sort?**
  - **Heapsort** & **Mergesort** have **O(NlogN)** worst-case running time.
  - **Quicksort** has **O(NlogN)** average-case running time

- **Theorem: Comparison sorting is $\Omega$(NlogN)**

**Cannot comparison-sort in linear time!**



## The Master Theorem

- Let **a** $\geq$ **1**, **b > 1**, **d** $\geq$ **0**, and **T(N)** be a monotonically increasing function of the form:
  - **$T(N) = aT(N/b) + O(N^d)$**;
    - **a** is the number of subproblems
    - **N/b** is the size of each subproblem
    - **$N^d$** is the "work done" to prepare the subproblems and assemble/combine the subresults
- Then:
  - **T(N) is $O(N^d)$; if $a < b^d$**
  - **T(N) is $O(N^d \log N)$; if $a = b^d$**
  - **T(N) is $O(N^{\log_b a})$; if $a > b^d$**

## Sorting in Linear Time?

### Yes! (but with non-comparison sort)

- **Condition:** if all values to be sorted are known to be integers between 1 and K (or any small range).
- **Bucket Sort Algorithm**:
  - Create an array of size K.
  - Put each element in its proper bucket.
  - If data is only integers, no need to store more than a count of how times that bucket has been used.
  - Output result via linear pass through array of buckets.

5

## Sorting in Linear Time?

- **Bucket Sort Algorithm**:

| `count` array | |
|---|---|
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |
| 4 | 2 |
| 5 | 3 |

**Example:**

If K=5 and for example:

**input:** (5,1,3,4,3,2,1,1,5,4,5)

**output:** (1,1,1,2,3,3,4,4,5,5,5)

6

## Bucket-Sort Algorithm. Analysis

- Overall running time complexity: ***O*(N+K)**
  - Linear in N, but also linear in K
  - $\Theta$(NlogN) lower bound does not apply because this is not a comparison sort
- Good method when K is smaller (or not much larger) than N
  - We don't spend time doing comparisons of duplicates
- Bad when K is much larger than N
  - Wasted space; wasted time during linear *O*(K) pass



## Radix-Sort

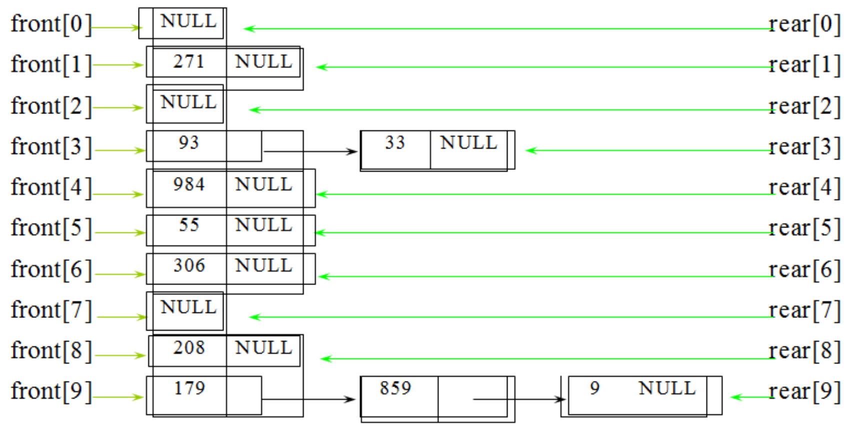
- Main Idea:
  - Use the **Bucket sort** on one digit at a time
  - Number of buckets = radix
  - Starting with Least Significant Digit (**LSD**)
  - Keeping sort stable
  - Do one pass per digit (to Most Significant Digit, **MSD**)
  - Invariant: After k passes (digits), the last k digits are sorted
- History: used in 1890 U.S. census by Hollerith (see URL below)

https://www.census.gov/history/www/innovations/technology/the_hollerith_tabulator.html

## Radix-Sort. Example

**d (digit) = 3, r (radix) = 10;**

**Input: 179, 208, 306, 93, 859, 984, 55, 9, 271, 33**

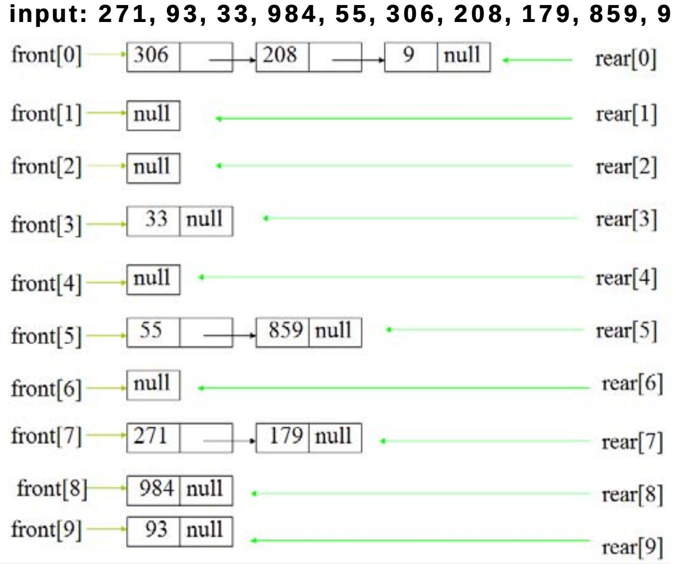



**After 1st Pass: 271, 93, 33, 984, 55, 306, 208, 179, 859, 9**

9

## Radix-Sort. Example (cont…)

**New input: 271, 93, 33, 984, 55, 306, 208, 179, 859, 9**

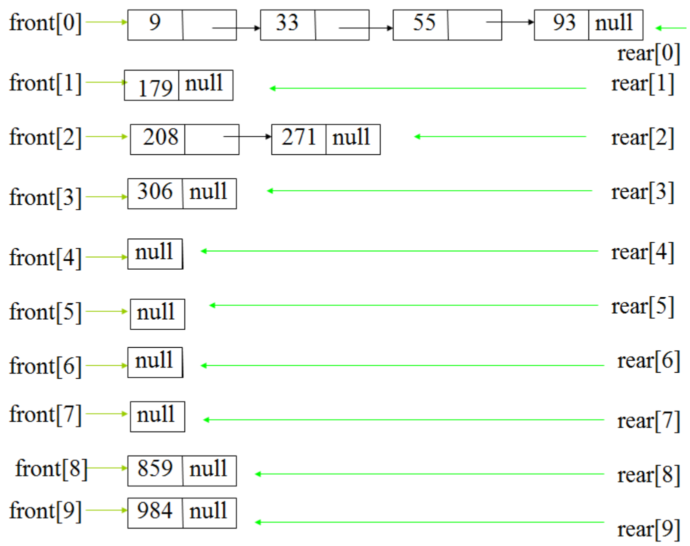



**After 2nd Pass: 306, 208, 9, 33, 55, 859, 271, 179, 984, 93**

10

## Radix-Sort. Example (cont…)

**New input: 306, 208, 9, 33, 55, 859, 271, 179, 984, 93**




**After 3st Pass: 9, 33, 55, 93, 179, 208, 271. 306, 859, 984** **Sorted!**

## Radix-Sort. Analysis

- **Input size:** Array of N elements
- **Number of buckets=** Radix: r
- **Number of passes** = # of "digits": d
- Work per pass is 1 bucket sort: O(r + N)
- **The running time complexity of RadixSort is O(d*(r+N))**
- Compared to comparison sorts, sometimes a win, but often not
  - **Example:** Strings of English letters (52 = 26 upper + 26 lower cases) up to length 15
    - Run-time proportional to: 15*(52 + N)
    - 15*(52 + N) < NlogN only if N > 33,000

12

## Summary on Sorting Algorithms

- Simple **$O(N^2)$** sorts can be fastest for small N.
  - **Selection sort** (not stable), **Insertion sort** (stable).
  - Used as "cut-off" to acelerate **Merge-Sort** and **Quick-Sort.**
- **Shell sort** – first **sub-quadratic, $O(N^{1.5})$, comparison sort** algorithm.
- **O(NLogN) comparison sort** algorithms:
  - **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.
- **Non-comparison sort algorithms:**
  - **Bucket-Sort** good for small number of possible key values.
  - **Radix-Sort** uses fewer buckets and more phases.

  https://www.youtube.com/watch?v=kPRA0W1kECg