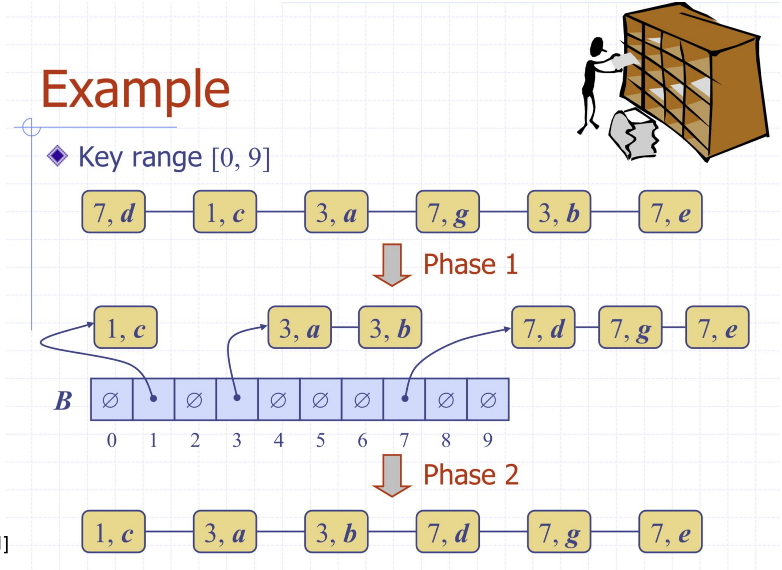
Radix Sort

- Stable Sorting Algorithms
- Lexicographic Order and Lexicographic Sorting
- Radix Sort Overview
- Examples
- Analysis

What is a **stable** sorting algorithm?

- Given items that we are sorting based on keys, a **stable** sorting algorithm will preserve the input ordering of equal keys
- Example: Single-key Bucket Sort (assuming items are added to the end of each list)



Picture from [1]

What about these?
- BubbleSort
- Insertion Sort
- Selection Sort
- HeapSort

Is Bubble Sort stable?

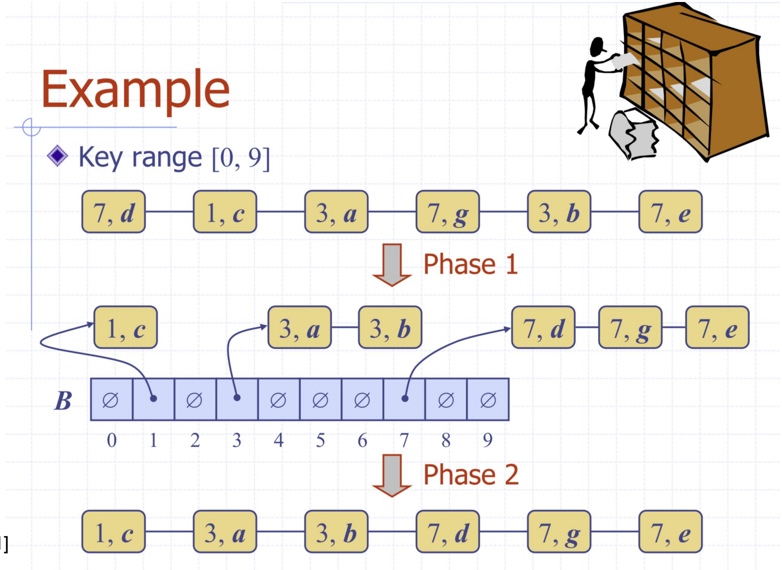
Is Selection Sort stable?

$$0 \quad \underline{2_1} \quad 2_2 \quad \underline{1}$$

$$0 \quad 1 \quad 2_2 \quad 2_1$$

What is a **stable** sorting algorithm?

- Given items that we are sorting based on keys, a **stable** sorting algorithms will preserve the input ordering of equal keys
- Example: Bucket Sort (assuming items are added to the end of each list)



Picture from [1]

What about these?

- BubbleSort--YES
- Insertion Sort--YES
- Selection Sort--NO
- HeapSort--NO

Lexicographic Order

- ***d***-tuple: a sequence of ***d*** keys $(k_1, k_2, \ldots, k_d)$
- k_i is called the i^{th} dimension of the tuple
- **lexicographic order:**

$$(x_1, x_2, \ldots, x_d) < (y_1, y_2, \ldots, y_d)$$

$$\Leftrightarrow$$

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

Picture from [1]

OR

Example 1: Are the following 4-tuples in lexicographic order?

(4, 7, 9, 1)
(4, 7, 9, 2)
(5, 3, 2, 0)
(5, 5, 5, 5)

YES

Example 2: Are the following 5-tuples in lexicographic order?

(0, 1, 2, 3, 4)
(1, 2, 3, 4, 5)
(1, 3, 4, 5, 5)
(1, 2, 3, 6, 6)

NO

Lexicographic Order

- ***d***-tuple: a sequence of ***d*** keys $(k_1, k_2, \ldots, k_d)$
- k_i is called the i^{th} dimension of the tuple
- **lexicographic order:**

$$(x_1, x_2, \ldots, x_d) < (y_1, y_2, \ldots, y_d)$$

$$\Leftrightarrow$$

$$x_1 < y_1 \lor x_1 = y_1 \land (x_2, \ldots, x_d) < (y_2, \ldots, y_d)$$

Picture from [1]

Example 1: Are the following 4-tuples in lexicographic order?

(4, 7, 9, 1)
(4, 7, 9, 2)
(5, 3, 2, 0)
(5, 5, 5, 5)

YES!

Example 2: Are the following 5-tuples in lexicographic order?

(0, 1, 2, 3, 4)
(1, 2, 3, 4, 5)
(1, 3, 4, 5, 5)
(1, 2, 3, 6, 6)

NO!

Lexicographic Sort

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)

Picture from [1]

Description: Sort the tuples by using a stable sorting method on each dimension—starting with the last dimension and working backwards.

WHY is it necessary to start from the last dimension and work backwards?

Least Significant First: the RIGHT way and the WRONG way

The Right Way

- (3, 5, 2) (3, 1, 8) (8, 3, 4) (3, 1, 6)
- (3, 5, **2**) (8, 3, **4**) (3, 1, **6**) (3, 1, **8**)
- (3, **1**, 6) (3, **1**, 8) (8, **3**, 4) (3, **5**, 2)
- (**3**, 1, 6) (**3**, 1, 8) (**3**, 5, 2) (**8**, 3, 4)

The Wrong Way

- (3, 5, 2) (3, 1, 8) (8, 3, 4) (3, 1, 6)
- (**3**, 5, 2) (**3**, 1, 8) (**3**, 1, 6) (**8**, 3, 4)
- (3, **1**, 8) (3, **1**, 6) (8, **3**, 4) (3, **5**, 2)
- (3, 5, **2**) (8, 3, **4**) (3, 1, **6**) (3, 1, **8**)

These are NOT in order!!!

Stable Sorting: the RIGHT way and the WRONG way

<u>The Right Way</u>

- (3, 5, 2) (3, 1, 8) (8, 3, 4) (3, 1, 6)
- (3, 5, **<u>2</u>**) (8, 3, **<u>4</u>**) (3, 1, **<u>6</u>**) (3, 1, **<u>8</u>**)
- (3, **<u>1</u>**, 6) (3, **<u>1</u>**, 8) (8, **<u>3</u>**, 4) (3, **<u>5</u>**, 2)
- (**<u>3</u>**, 1, 6) (**<u>3</u>**, 1, 8) (**<u>3</u>**, 5, 2) (**<u>8</u>**, 3, 4)

<u>The Wrong Way</u>

- (3, 5, 2) (3, 1, 8) (8, 3, 4) (3, 1, 6)
- (3, 5, **<u>2</u>**) (8, 3, **<u>4</u>**) (3, 1, **<u>6</u>**) (3, 1, **<u>8</u>**)
- (3, **<u>1</u>**, 8) (3, **<u>1</u>**, 6) (8, **<u>3</u>**, 4) (3, **<u>5</u>**, 2)
- (**<u>3</u>**, 5, 2) (**<u>3</u>**, 1, 6) (**<u>3</u>**, 1, 8) (**<u>8</u>**, 3, 4)

<u>These are NOT in order!!!</u>

A couple things...

- Why is it important to use a stable sorting method when sorting each dimension?

It's important to maintain the relative order of equal elements so that the previously sorted dimensions remain sorted.

- Why is it important to sort from least significant to most significant dimension?

It has to do with the way lexicographic order is defined (from left to right). Therefore, to maintain that order, the sorting needs to happen from right to left so that later sorts don't "undo" previous sorts. (i.e. you have to start with the "least significant" item first.)

Runtime Analysis

Given **N** tuples of **d** dimensions and a stable sorting algorithm that sorts **N** elements in time **T(N)**, what is the runtime of a lexicographic sorting algorithm?

$$\underbrace{\text{Sorting } \underset{T(N)}{N} \text{ elements } d \text{ times}}$$

$\boxed{dT(N)}$

$O(dT(N))$

Runtime Analysis

Given **N** tuples of **d** dimensions and a stable sorting algorithm that sorts **N** elements in time **T(N)**, what is the runtime of a lexicographic sorting algorithm?

Basically we are running the stable sorting algorithm on **N** elements **d** times: **O(dT(N))**

Radix Sort

Algorithm *radixSort(S, N)*

Input sequence S of d-tuples such that $(0, ..., 0) \leq (x_1, ..., x_d)$ and $(x_1, ..., x_d) \leq (k-1, ...k-1)$ for each tuple $(x_1, ..., x_d)$ in S

Output sequence S sorted in lexicographic order

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

  bucketSort the tuples using the i^{th} dimension as the keys

- Lexicographic Sort where **the stable sort is Bucket Sort**
- Applicable when the keys in each dimension are integers in [0, k - 1]
- Runtime?

Radix Sort

Algorithm *radixSort*(S, N)

Input sequence S of d-tuples such that $(0, ..., 0) \leq (x_1, ..., x_d)$ and $(x_1, ..., x_d) \leq (k-1, ...k-1)$ for each tuple $(x_1, ..., x_d)$ in S

Output sequence S sorted in lexicographic order

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

bucketSort the tuples using the i^{th} dimension as the keys

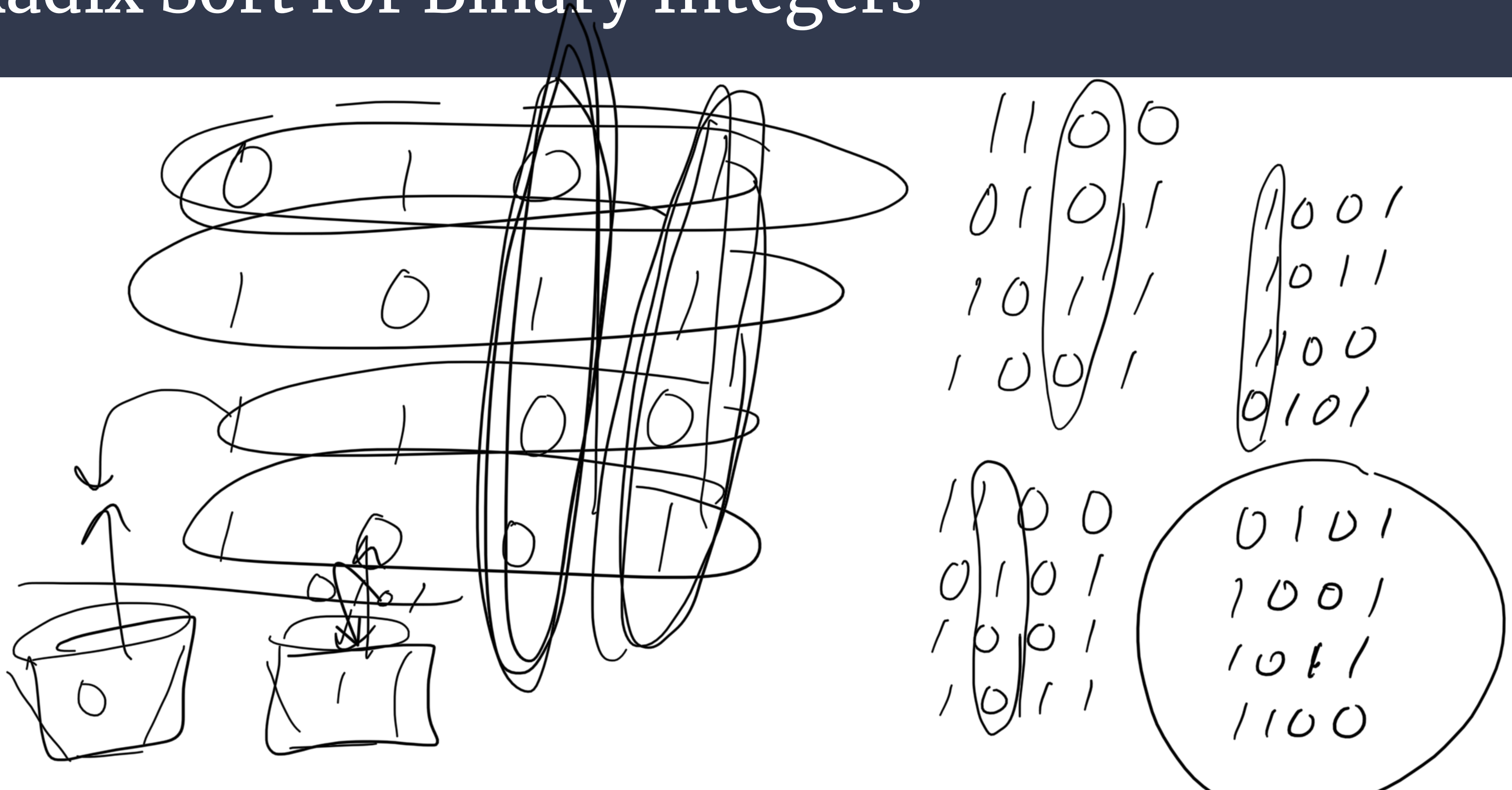
- Lexicographic Sort where **the stable sort is Bucket Sort**
- Applicable when the keys in each dimension are integers in [0, k - 1]
- Runtime?

Assuming the version of Bucket Sort where each bucket only has one possible key-value…

O(d(N + k))

where there are **N** tuples of **d** dimensions and **k** buckets.

Radix Sort for Binary Integers



Radix Sort for Binary Integers

4

$4 \Rightarrow 100$

- **Input**: A sequence of **N** ***b***-bit integers (e.g. $x = x_{b-1}...x_1x_0$)
- **Overview**: Treat each integer like a tuple with **b** dimensions and sort with radix sort
- The **key range** (for buckets) is [0, 1], so **k = 2**
- **Analysis:** Runs in **O(b(N + 2)) = O(N)** if **b** is a constant
- **What does this mean?**

$32(N+2) \Rightarrow O(N)$

We can sort (for example) a sequence of 32-bit integers in LINEAR TIME!!!

Algorithm *binaryRadixSort*(S)

Input sequence S of b-bit integers

Output sequence S sorted

replace each element x of S with the item $(0, x)$

for $i \leftarrow 0$ **to** $b - 1$

replace the key k of each item (k, x) of S with bit x_i of x

bucketSort(S, 2)

Picture from [1]

A Few More Things about Sorting...

- **Comparison-based Sorting:** uses a ***comparator*** that determines the order of the sorted list (i.e. whether item A or item B should occur first in the final sorted list)
- **Examples of comparison-based sorting algorithms:** BubbleSort, Selection Sort, Insertion Sort, ShellSort, HeapSort, MergeSort, QuickSort…
- **Examples of non-comparison-based sorting algorithms:** BucketSort (though it sometimes uses one to sort the buckets), Radix Sort
- The **proven lower bound** for comparison-based sorting (in the worst-case) is **Ω(NlogN).**

Counting Sort

References

[1] Tamassia and Goodrich