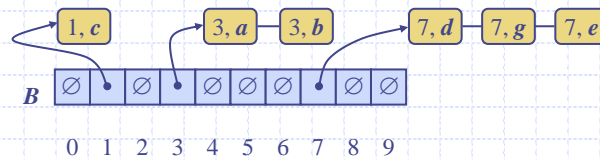


# Lecture 10

## Bucket Sort & Radix-Sort



Bucket-Sort and Radix-Sort

1

## Bucket-Sort (§ 4.5.1)



- ◆ 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 item  $(k, o)$  into its bucket  $B[k]$
    - Phase 2: For  $i = 0, \dots, N - 1$ , move the items 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*( $S, N$ )

**Input** sequence  $S$  of (key, element) items with keys in the range  $[0, N - 1]$

**Output** sequence  $S$  sorted by increasing keys

$B \leftarrow$  array of  $N$  empty sequences

**while**  $\neg S.isEmpty()$

$f \leftarrow S.first()$

$(k, o) \leftarrow S.remove(f)$

$B[k].insertLast((k, o))$

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

**while**  $\neg B[i].isEmpty()$

$f \leftarrow B[i].first()$

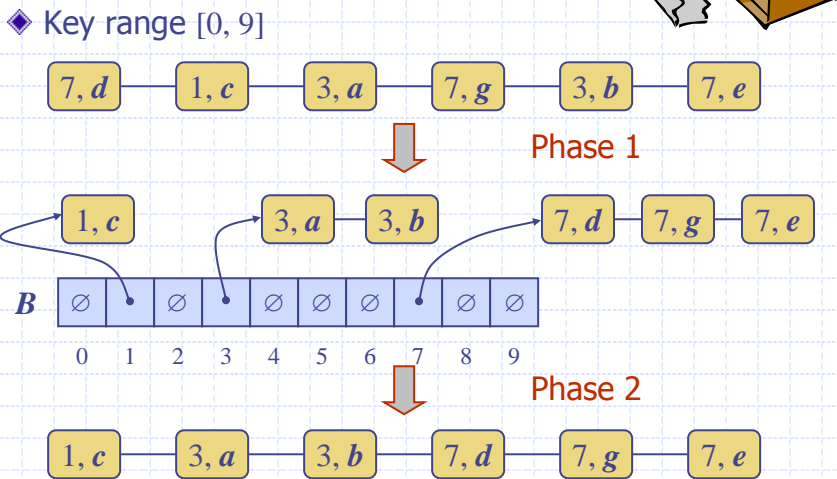
$(k, o) \leftarrow B[i].remove(f)$

$S.insertLast((k, o))$

Bucket-Sort and Radix-Sort

2

# Example



Bucket-Sort and Radix-Sort

# 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 item  $(k, o)$  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 item  $(k, o)$  into bucket  $B[r(k)]$

Bucket-Sort and Radix-Sort

## Lexicographic Order



- ◆ A  $d$ -tuple is a sequence of  $d$  keys  $(k_1, k_2, \dots, k_d)$ , 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

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



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

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:

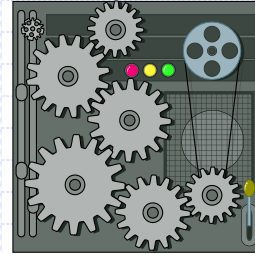
(7,4,6) (5,1,5) (2,4,6) (2, 1, 4) (3, 2, 4)

(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 (§ 4.5.2)



- ◆ 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  $O(d(n + N))$

### Algorithm *radixSort*( $S, N$ )

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

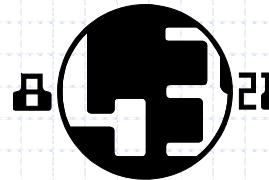
**Output** sequence  $S$  sorted in lexicographic order

**for**  $i \leftarrow d$  **downto** 1  
    *bucketSort*( $S, N$ )

Bucket-Sort and Radix-Sort

7

## Radix-Sort for Binary Numbers



- ◆ Consider a sequence of  $n$   $b$ -bit integers  

$$x = x_{b-1} \dots x_1 x_0$$
- ◆ We represent each element as a  $b$ -tuple of integers in the range  $[0, 1]$  and apply radix-sort with  $N = 2$
- ◆ This application of the radix-sort algorithm runs in  $O(bn)$  time
- ◆ For example, we can sort 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*( $S, 2$ )

Bucket-Sort and Radix-Sort

8

# Example



◆ Sorting a sequence of 4-bit integers

