

# CSc445 Algorithms

## Sorting in linear time.

Slides curacy of  
Piotr Indyk and Carola Wenk

## Sorting in linear time

### Counting sort:

Applicable when all elements are **integers**, in a small range, say between  $l$ , and  $k$ .

- **Input:**  $A[1 \dots n]$ , where  $A[j] \in \{1, 2, \dots, k\}$ , for every  $j$ .
- **Output:**  $B[1 \dots n]$ , sorted
- **Auxiliary storage:**  $C[1 \dots k]$ .

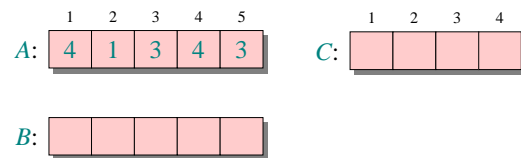
## Counting sort

```

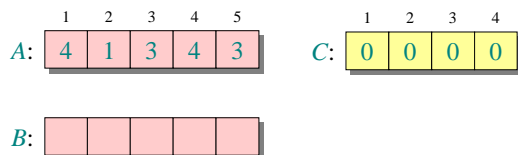
for  $i \leftarrow 1$  to  $k$ 
  do  $C[i] \leftarrow 0$ 
for  $j \leftarrow 1$  to  $n$ 
  do  $C[A[j]]++$ 
  /* After this step,  $C[i]$  contains the number of elements
  that equal  $i$ , that is  $C[i] = |\{j | A[j] = i\}|$  */
for  $i \leftarrow 2$  to  $k$ 
  do  $C[i] \leftarrow C[i] + C[i-1]$ 
  /* After this step  $C[i] = |\{j | A[j] \leq i\}|$  */
  /* We are about to scan  $A$ .
  /* Invariant:  $C[x]$  points to cell of  $B$  into which we should
  copy the next  $x$  we encounter */
for  $j \leftarrow n$  downto  $1$ 
  do  $x \leftarrow A[j]$ 
   $B[C[x]] \leftarrow x$ 
   $C[x]--$ 

```

## Counting-sort example



## Loop 1

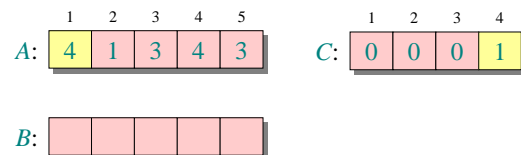


```

for  $i \leftarrow 1$  to  $k$ 
  do  $C[i] \leftarrow 0$ 

```

## Loop 2



```

for  $j \leftarrow 1$  to  $n$ 
  do  $C[A[j]] \leftarrow C[A[j]] + 1$   $C[i] = |\{\text{key} = i\}|$ 

```

### Loop 2

A: 

1	2	3	4	5
4	1	3	4	3

 C: 

1	2	3	4
1	0	0	1

B: 

--	--	--	--	--

for  $j \leftarrow 1$  to  $n$   
do  $C[A[j]] \leftarrow C[A[j]] + 1$  □  $C[i] = |\{\text{key} = i\}|$

### Loop 2

A: 

1	2	3	4	5
4	1	3	4	3

 C: 

1	2	3	4
1	0	1	1

B: 

--	--	--	--	--

for  $j \leftarrow 1$  to  $n$   
do  $C[A[j]] \leftarrow C[A[j]] + 1$  □  $C[i] = |\{\text{key} = i\}|$

### Loop 2

A: 

1	2	3	4	5
4	1	3	4	3

 C: 

1	2	3	4
1	0	1	2

B: 

--	--	--	--	--

for  $j \leftarrow 1$  to  $n$   
do  $C[A[j]] \leftarrow C[A[j]] + 1$  □  $C[i] = |\{\text{key} = i\}|$

### Loop 2

A: 

1	2	3	4	5
4	1	3	4	3

 C: 

1	2	3	4
1	0	2	2

B: 

--	--	--	--	--

for  $j \leftarrow 1$  to  $n$   
do  $C[A[j]] \leftarrow C[A[j]] + 1$  □  $C[i] = |\{\text{key} = i\}|$

### Loop 3

A: 

1	2	3	4	5
4	1	3	4	3

 C: 

1	2	3	4
1	0	2	2

B: 

--	--	--	--	--

 C': 

1	1	2	2
---	---	---	---

for  $i \leftarrow 2$  to  $k$   
do  $C[i] \leftarrow C[i] + C[i-1]$  □  $C[i] = |\{\text{key} \leq i\}|$

### Loop 3

A: 

1	2	3	4	5
4	1	3	4	3

 C: 

1	2	3	4
1	0	2	2

B: 

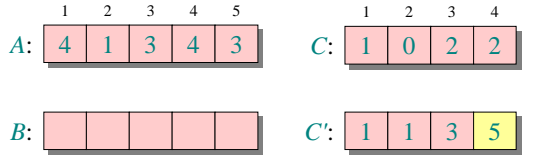
--	--	--	--	--

 C': 

1	1	3	2
---	---	---	---

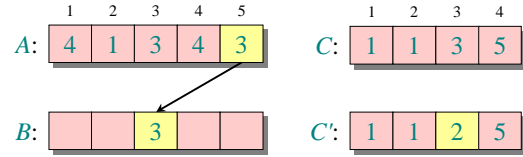
for  $i \leftarrow 2$  to  $k$   
do  $C[i] \leftarrow C[i] + C[i-1]$  □  $C[i] = |\{\text{key} \leq i\}|$

### Loop 3



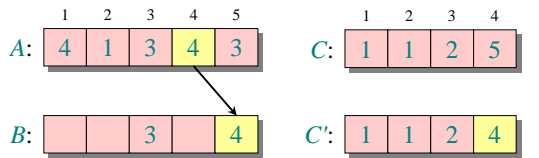
for  $i \leftarrow 2$  to  $k$   
do  $C[i] \leftarrow C[i] + C[i-1]$      $\square C[i] = |\{\text{key} \leq i\}|$

### Loop 4



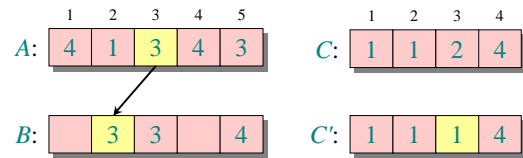
for  $j \leftarrow n$  downto 1  
do  $x \leftarrow A[j]$   
   $B[C[x]] \leftarrow x$   
   $C[x] \leftarrow C[x] -$

### Loop 4



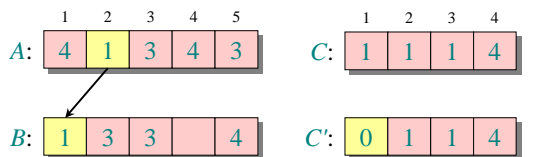
for  $j \leftarrow n$  downto 1  
do  $x \leftarrow A[j]$   
   $B[C[x]] \leftarrow x$   
   $C[x] \leftarrow C[x] -$

### Loop 4



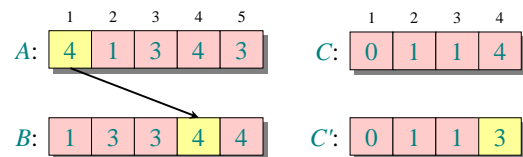
for  $j \leftarrow n$  downto 1  
do  $x \leftarrow A[j]$   
   $B[C[x]] \leftarrow x$   
   $C[x] \leftarrow C[x] -$

### Loop 4



for  $j \leftarrow n$  downto 1  
do  $x \leftarrow A[j]$   
   $B[C[x]] \leftarrow x$   
   $C[x] \leftarrow C[x] -$

### Loop 4



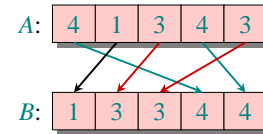
for  $j \leftarrow n$  downto 1  
do  $x \leftarrow A[j]$   
   $B[C[x]] \leftarrow x$   
   $C[x] \leftarrow C[x] -$

## Analysis

$$\begin{array}{l}
 \Theta(k) \left\{ \begin{array}{l} \text{for } i \leftarrow 1 \text{ to } k \\ \text{do } C[i] \leftarrow 0 \end{array} \right. \\
 \Theta(n) \left\{ \begin{array}{l} \text{for } j \leftarrow 1 \text{ to } n \\ \text{do } C[A[j]] \leftarrow C[A[j]] + 1 \end{array} \right. \\
 \Theta(k) \left\{ \begin{array}{l} \text{for } i \leftarrow 2 \text{ to } k \\ \text{do } C[i] \leftarrow C[i] + C[i-1] \end{array} \right. \\
 \Theta(n) \left\{ \begin{array}{l} \text{for } j \leftarrow n \text{ downto } 1 \\ \text{do } x \leftarrow A[j] \\ \quad B[C[x]] \leftarrow x \\ \quad C[x] -- \end{array} \right. \\
 \hline
 \Theta(n+k)
 \end{array}$$

## Stable sorting

Counting sort is a **stable** sort: it preserves the input order among equal elements.



**Exercise:** What other sorts have this property?

## Radix sort

- Used (for example) to sort integers in the range 0 to 10000.
- In general – good for any **lexicographic** sorting.
- **Origin:** Herman Hollerith's card-sorting machine for the 1890 U.S. Census. (See Appendix.)
- Digit-by-digit sort.
- Hollerith's original (bad) idea: sort on most-significant digit first.
- Good idea: Sort on **least-significant digit first** with auxiliary **stable** sort.

## Terminology

We sort **records**:

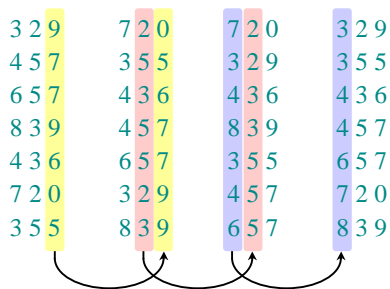
We distinguish between the

- **Key** (e.g. student i.d.)
- **Auxiliary data** (e.g. the student's address)

Of course, when copying data during sorting, we copy the whole record.

**Stable sorting:** The order between two records with the same key is not changed.

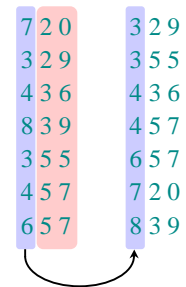
## Operation of radix sort



## Correctness of radix sort

*Induction on digit position*

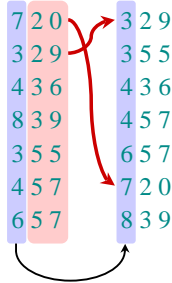
- Assume that the numbers are sorted by their low-order  $t-1$  digits.
- Sort on digit  $t$



## Correctness of radix sort

Induction on digit position

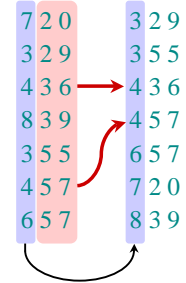
- Assume that the numbers are sorted by their low-order  $t - 1$  digits.
- Sort on digit  $t$ 
  - Two numbers that differ in digit  $t$  are correctly sorted.



## Correctness of radix sort

Induction on digit position

- Assume that the numbers are sorted by their low-order  $t - 1$  digits.
- Sort on digit  $t$ 
  - Two numbers that differ in digit  $t$  are correctly sorted.
  - Two numbers equal in digit  $t$  are put in the same order as the input  $\Rightarrow$  correct order.



## Analysis of radix sort

- Assume counting sort is the auxiliary stable sort.
- Sort  $n$  computer words of  $b$  bits each.
- Each word can be viewed as having  $b/r$  base- $2^r$  digits.

**Example:**  $b=32$ -bit word 

$r=8$	8	8	8
-------	---	---	---

$r = 8 \Rightarrow b/r = 4$  passes of counting sort on base- $2^8$  digits; or  $r = 16 \Rightarrow b/r = 2$  passes of counting sort on base- $2^{16}$  digits.

*How many passes should we make?*

## Analysis (continued)

**Recall:** Counting sort takes  $\Theta(n + k)$  time to sort  $n$  numbers in the range from 0 to  $k - 1$ .

If each  $b$ -bit word is broken into  $r$ -bit pieces, each pass of counting sort takes  $\Theta(n + 2^r)$  time. Since there are  $b/r$  passes, we have

$$T(n, b) = \Theta\left(\frac{b}{r}(n + 2^r)\right).$$

Choose  $r$  to minimize  $T(n, b)$ :

- Increasing  $r$  means fewer passes, but as  $r \gg \lg n$ , the time grows exponentially.

## Choosing $r$

$$T(n, b) = \Theta\left(\frac{b}{r}(n + 2^r)\right)$$

Minimize  $T(n, b)$  by differentiating and setting to 0.

Or, just observe that we don't want  $2^r \gg n$ , and there's no harm asymptotically in choosing  $r$  as large as possible subject to this constraint.

Choosing  $r = \lg n$  implies  $T(n, b) = \Theta(bn \lg n)$ .

- For numbers in the range from 0 to  $n^d - 1$ , we have  $b = d \lg n \Rightarrow$  radix sort runs in  $\Theta(dn)$  time.

## Combining Radix and Merge Sorts

$\{(x_i, y_i) \dots (x_n, y_n)\}$  are **lexicographically** sorted if for every  $i$ , either  $x_i < x_{i+1}$  or  $x_i = x_{i+1}$  and  $y_i \leq y_{i+1}$ .

**Example:** input  $n$  records of students: For each student we store his/hers family name and first name.

Algorithm: Sort first by first name, and then by family name, using a stable sorting algorithm.

MergeSort is stable.

## Conclusions

In practice, radix sort is fast for large inputs, as well as simple to code and maintain.

**Example** (32-bit numbers):

- At most 3 passes when sorting  $\geq 2000$  numbers.
- Merge sort and quicksort do at least  $\lceil \lg 2000 \rceil = 11$  passes.

**Downside:** Unlike quicksort, radix sort displays little locality of reference, and thus a well-tuned quicksort fares better on modern processors, which feature steep memory hierarchies.

## Origin of radix sort

Hollerith's original 1889 patent alludes to a most-significant-digit-first radix sort:

*"The most complicated combinations can readily be counted with comparatively few counters or relays by first assorting the cards according to the first items entering into the combinations, then reassorting each group according to the second item entering into the combination, and so on, and finally counting on a few counters the last item of the combination for each group of cards."*

Least-significant-digit-first radix sort seems to be a folk invention originated by machine operators.

## Herman Hollerith (1860-1929)



- The 1880 U.S. Census took almost 10 years to process.
- While a lecturer at MIT, Hollerith prototyped punched-card technology.
- His machines, including a "card sorter," allowed the 1890 census total to be reported in 6 weeks.
- He founded the Tabulating Machine Company in 1911, which merged with other companies in 1924 to form International Business Machines.

## Punched cards

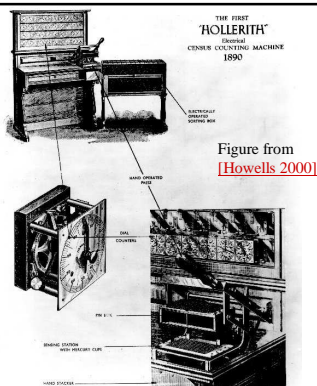
- Punched card = data record.
- Hole = value.
- Algorithm = machine + human operator.



Replica of punch card from the 1900 U.S. census. [Howells 2000]

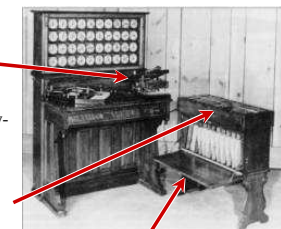
## Hollerith's tabulating system

- Pantograph card punch
- Hand-press reader
- Dial counters
- Sorting box



## Appendix: Operation of Hollerith sorter

- An operator inserts a card into the press.
- Pins on the press reach through the punched holes to make electrical contact with mercury-filled cups beneath the card.
- Whenever a particular digit value is punched, the lid of the corresponding sorting bin lifts.
- The operator deposits the card into the bin and closes the lid.
- When all cards have been processed, the front panel is opened, and the cards are collected in order, yielding one pass of a stable sort.



Hollerith Tabulator, Pantograph, Press, and Sorter

