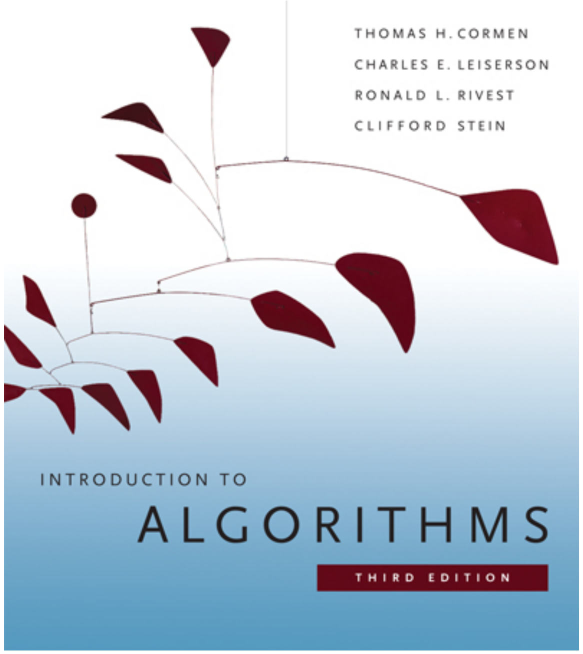
6.006- *Introduction to Algorithms*



Lecture 10

Prof. Constantinos Daskalakis

CLRS 8.1-8.4

Menu

- Show that $\Theta(n \lg n)$ is the best possible running time for a sorting algorithm.
- Design an algorithm that sorts in $\Theta(n)$ time.
- Hint: maybe the models are different ?

Comparison sort

All the sorting algorithms we have seen so far are ***comparison sorts***: only use comparisons to determine the relative order of elements.

- *E.g.*, merge sort, heapsort.

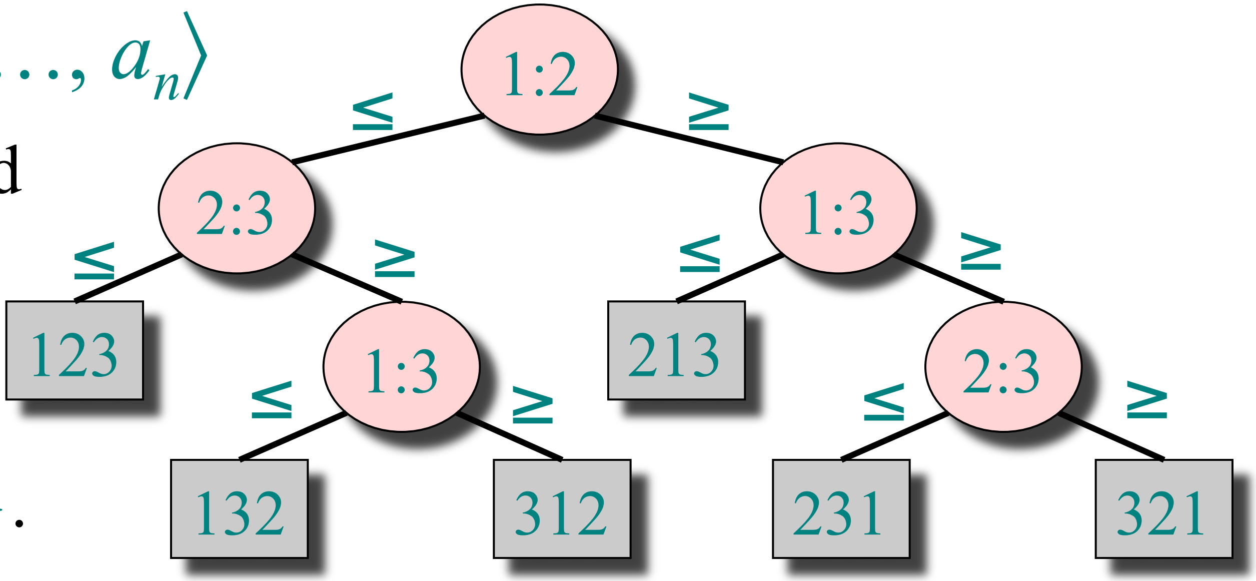
The best running time that we've seen for comparison sorting is $O(n \lg n)$.

Is $O(n \lg n)$ the best we can do?

Decision trees can help us answer this question.

Decision-tree

A recipe for sorting n numbers $\langle a_1, a_2, \ldots, a_n \rangle$



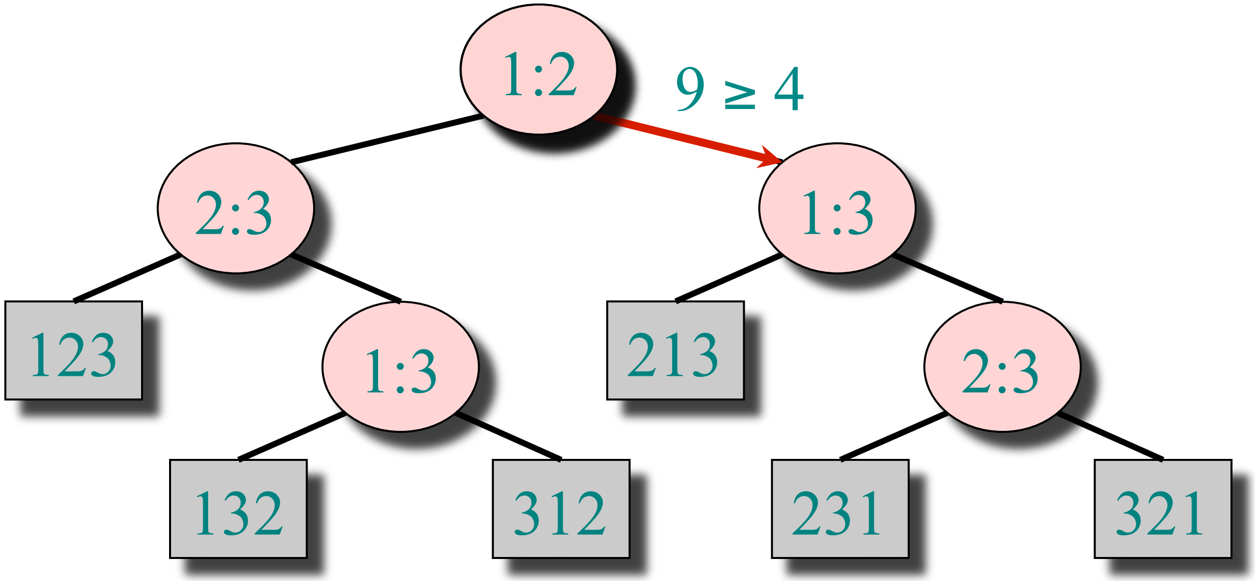
- Nodes are suggested comparisons: $i{:}j$ means compare a_i to a_j, for $i, j \in \{1, 2, \ldots, n\}$.

- Branching direction depends on outcome of comparisons.

- Leaves are labeled with permutations corresponding to the outcome of the sorting.

Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle$
$= \langle\ 9, 4, 6\ \rangle$:

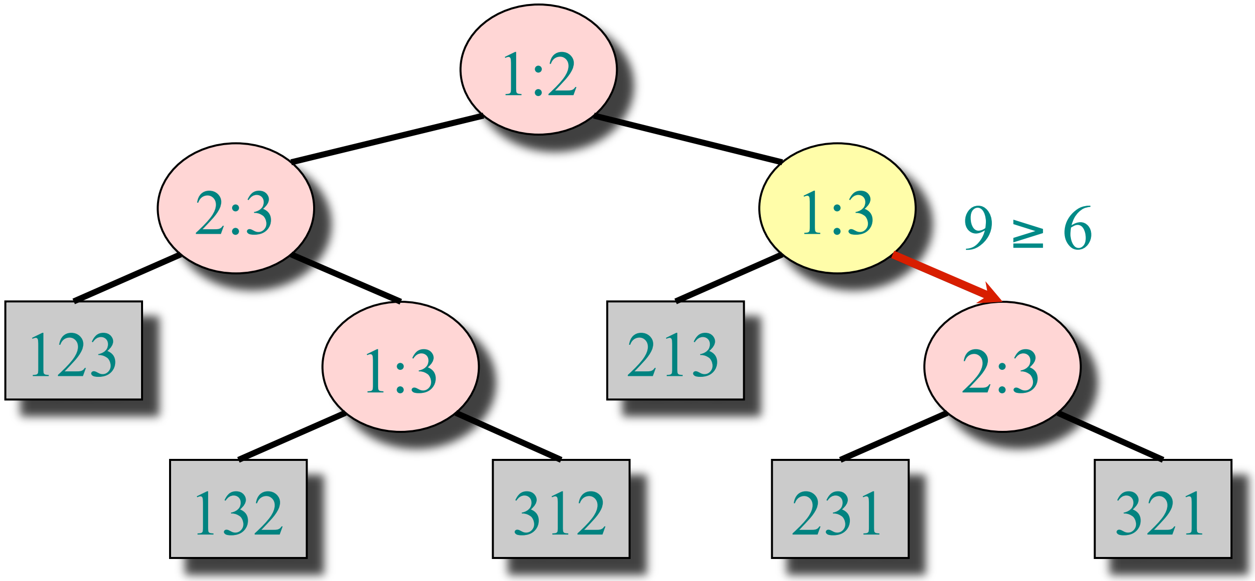


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2,\ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \le a_j$.
- The right subtree shows subsequent comparisons if $a_i \ge a_j$.

Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle$ $= \langle\ 9, 4, 6\ \rangle$:

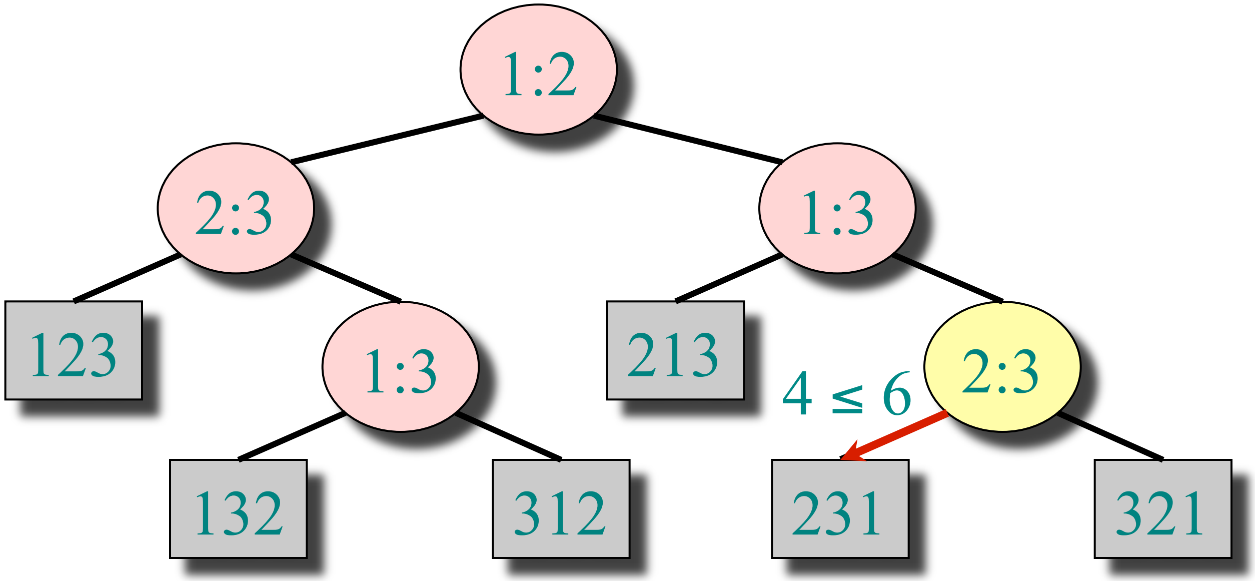


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \le a_j$.
- The right subtree shows subsequent comparisons if $a_i \ge a_j$.

Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle$
$= \langle$ 9, 4, 6 $\rangle$:

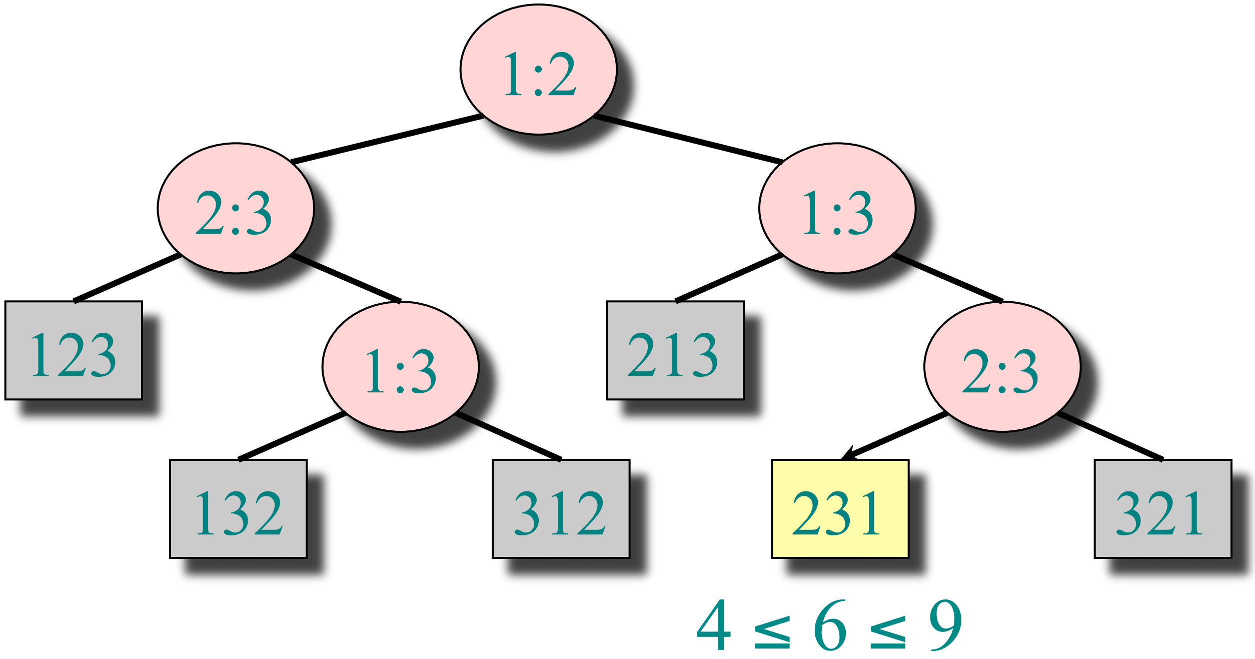


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \le a_j$.
- The right subtree shows subsequent comparisons if $a_i \ge a_j$.

Decision-tree example

Sort $\langle a_1, a_2, a_3\rangle$ $= \langle\ 9, 4, 6\ \rangle$:



Each leaf contains a permutation $\langle\pi(1), \pi(2),\ldots, \pi(n)\rangle$ to indicate that the ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$ has been established.

Decision-tree model

A decision tree can model the execution of any comparison sort:

- One tree for each input size n.
- A path from the root to the leaves of the tree represents a trace of comparisons that the algorithm may perform.
- The running time of the algorithm = the length of the path taken.
- Worst-case running time = height of tree.

Lower bound for decision-tree sorting

Theorem. Any decision tree that can sort n elements must have height $\Omega(n \lg n)$.

Proof. (Hint: how many leaves are there?)

- The tree must contain $\geq n!$ leaves, since there are $n!$ possible permutations
- A height-h binary tree has $\leq 2^h$ leaves
- Thus

$$\begin{aligned} 2^h &\geq n! \\ h &\geq \lg(n!) && \text{(lg is mono. increasing)} \\ &\geq \lg((n/e)^n) && \text{(Stirling's formula)} \\ &= n \lg n - n \lg e \\ &= \Omega(n \lg n)\,. \end{aligned}$$

Sorting in linear time

Counting sort: No comparisons between elements.

- ***Input***: $A[1 \,.\,.\, n]$, where $A[j] \in \{1, 2, \ldots, k\}$.
- ***Output***: $B[1 \,.\,.\, n]$, a sorted permutation of A
- ***Auxiliary storage***: $C[1 \,.\,.\, k]$.

Counting sort

```
for $i \leftarrow 1$ to $k$
    do $C[i] \leftarrow 0$
for $j \leftarrow 1$ to $n$
    do $C[A[j]] \leftarrow C[A[j]] + 1$
```

store in C the frequencies of the different keys in A
i.e. $C[i] = |\{\text{key} = i\}|$

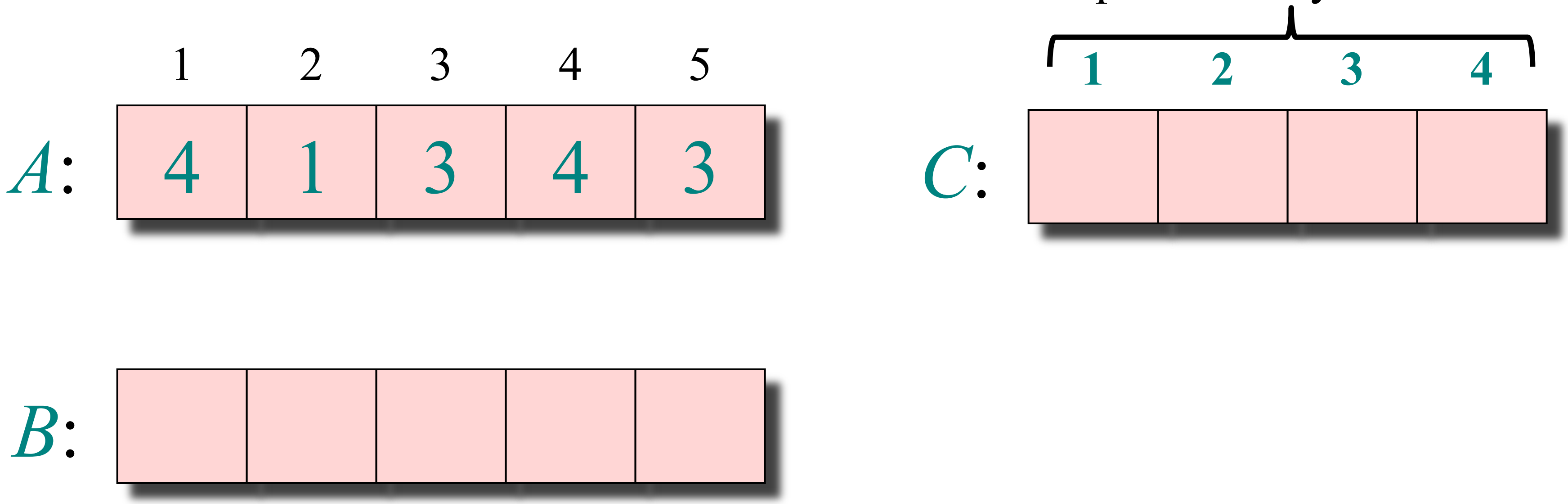
```
for $i \leftarrow 2$ to $k$
    do $C[i] \leftarrow C[i] + C[i-1]$
```

now C contains the cumulative frequencies of different keys in A, i.e. $C[i] = |\{\text{key} \le i\}|$

```
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

using cumulative frequencies build sorted permutation

Counting-sort example



one index for each possible key stored in A

Loop 1: initialization

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	**2**	**3**	**4**
C:	0	0	0	0

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

Loop 2: count frequencies

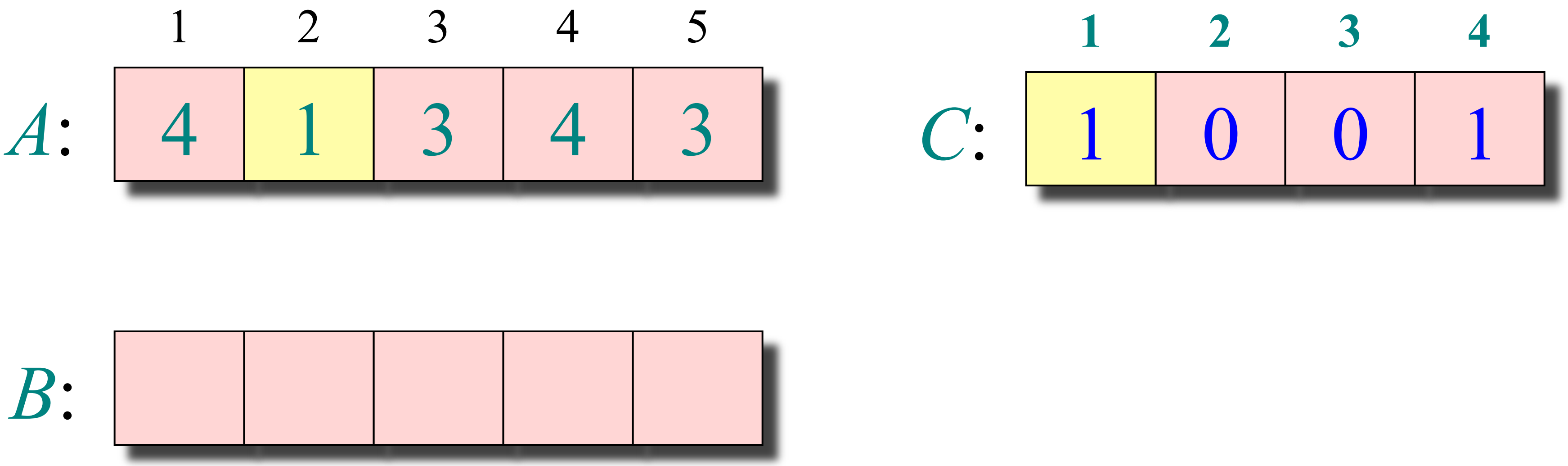
	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	0	0	0	1

B:					

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

Loop 2: count frequencies



	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	0	1

B:					

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

Loop 2: count frequencies

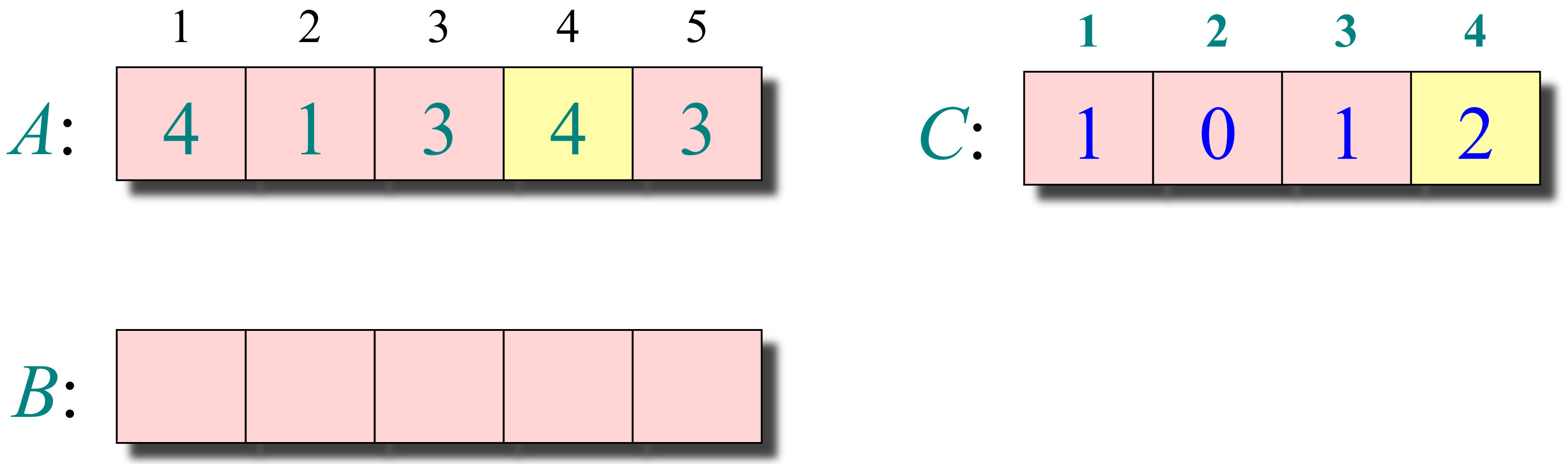
	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	1	1

B:					

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

Loop 2: count frequencies



	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	1	2

B:					

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

Loop 2: count frequencies

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	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 2: count frequencies

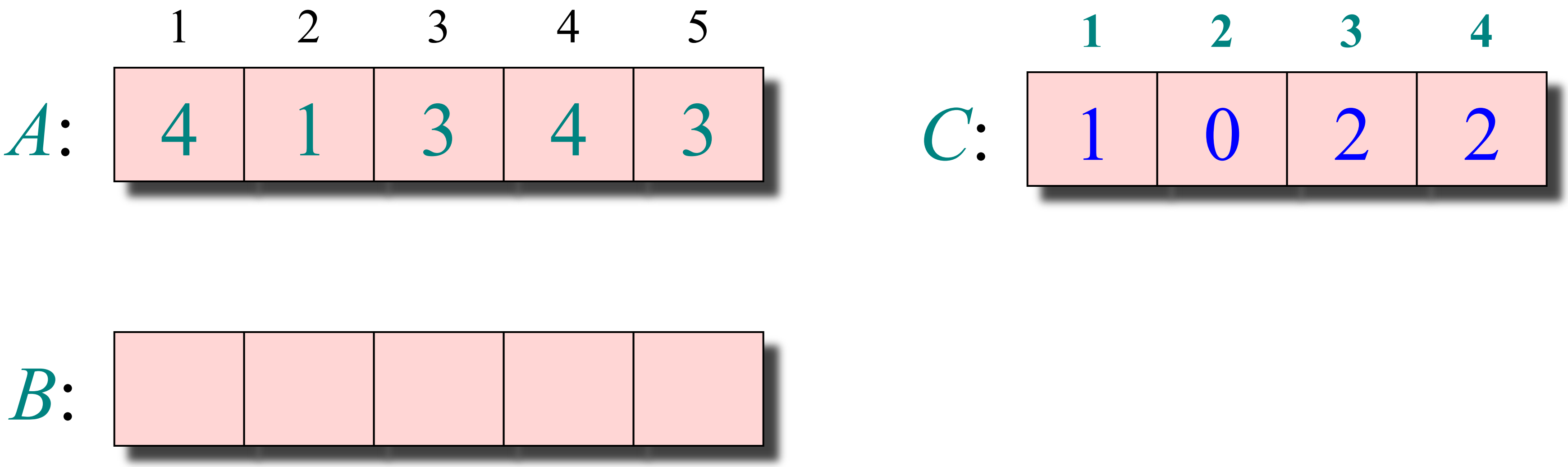
	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	2	2

B:					

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

[A parenthesis: a quick finish



	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	2	2

B:					

Walk through frequency array an place the appropriate number of each key in output array…

A parenthesis: a quick finish

	1	2	3	4	5
A:	4	1	3	4	3
B:	1				

	1	**2**	**3**	**4**
C:	1	0	2	2

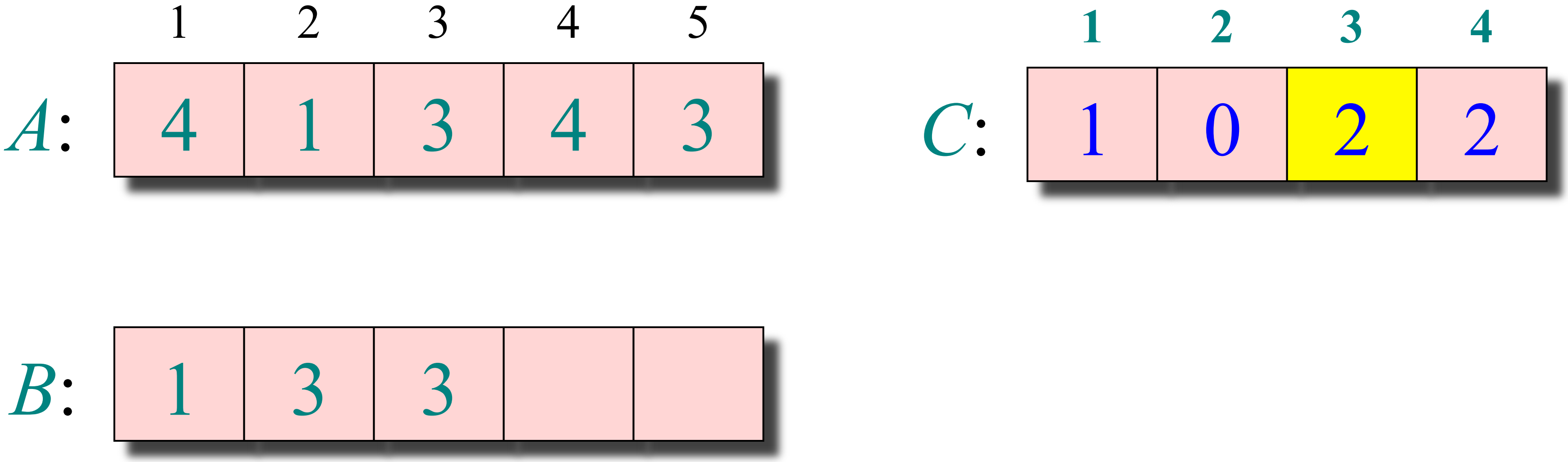
A parenthesis: a quick finish

	1	2	3	4	5
A:	4	1	3	4	3

B:	1				

	1	**2**	**3**	**4**
C:	1	0	2	2

A parenthesis: a quick finish



	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	2	2

B:	1	3	3		

A parenthesis: a quick finish

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	2	2

B:	1	3	3	4	4

B is sorted!

but it is not “stably sorted”…]

Loop 2: count frequencies

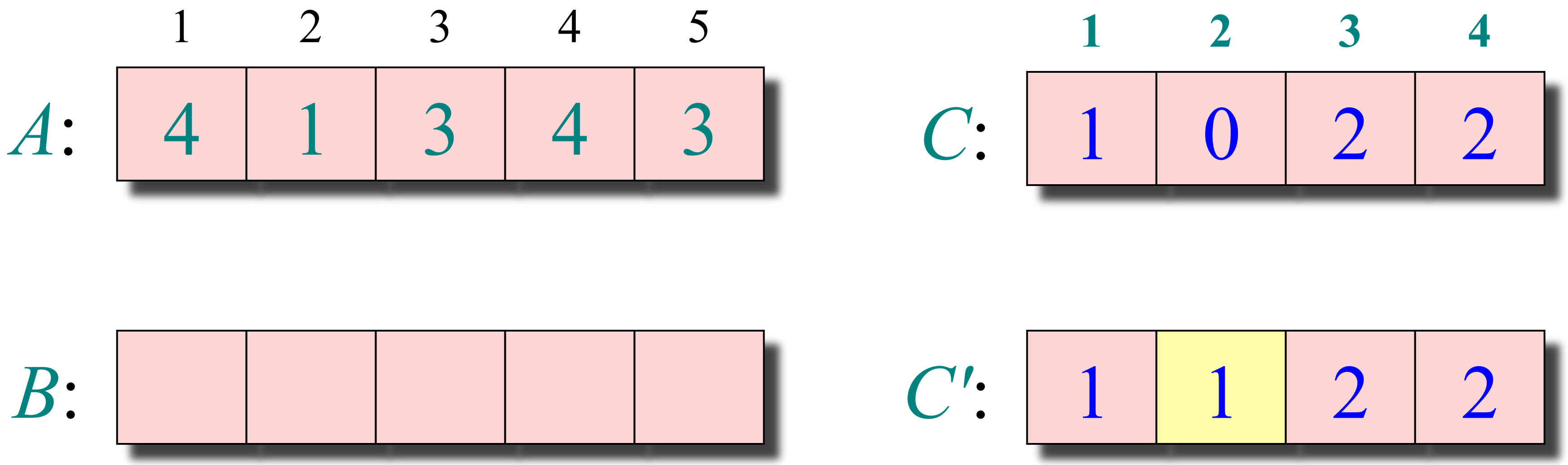
	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	2	2

B:					

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

Loop 3: cumulative frequencies



	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	1	0	2	2
C':	1	1	2	2

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

Loop 3: cumulative frequencies

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	1	0	2	2
C':	1	1	3	2

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

Loop 3: cumulative frequencies

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	**2**	**3**	**4**
C:	1	0	2	2
C':	1	1	3	5

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

Loop 4: permute elements of A

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	1	3	5

B:					

```
for $j \leftarrow n$ downto 1
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4: permute elements of A

	1	2	3	4	5
A:	4	1	3	4	3

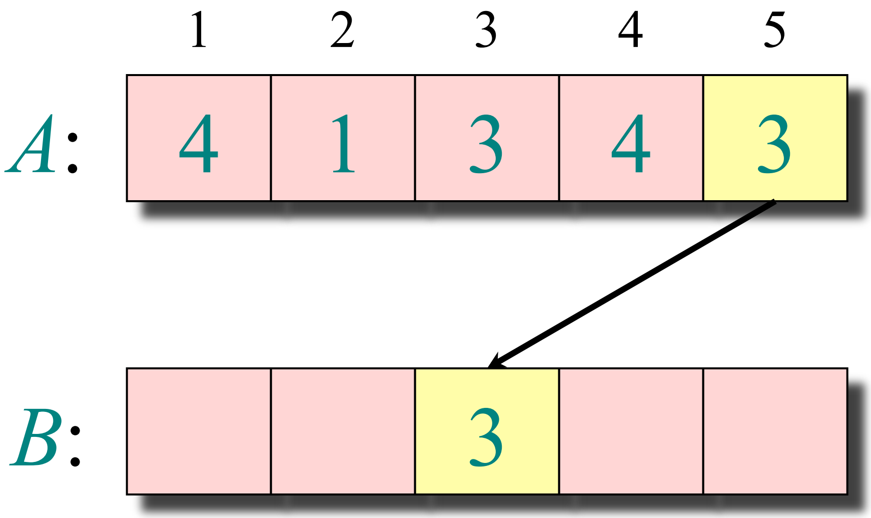
	1	2	3	4
C:	1	1	3	5

B:					

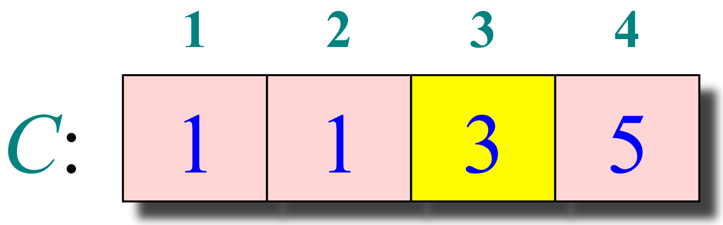
There are exactly 3 elements ≤A[5]; so where should I place A[5]?

```
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4: permute elements of A



	1	2	3	4	5
A:	4	1	3	4	3
B:			3		

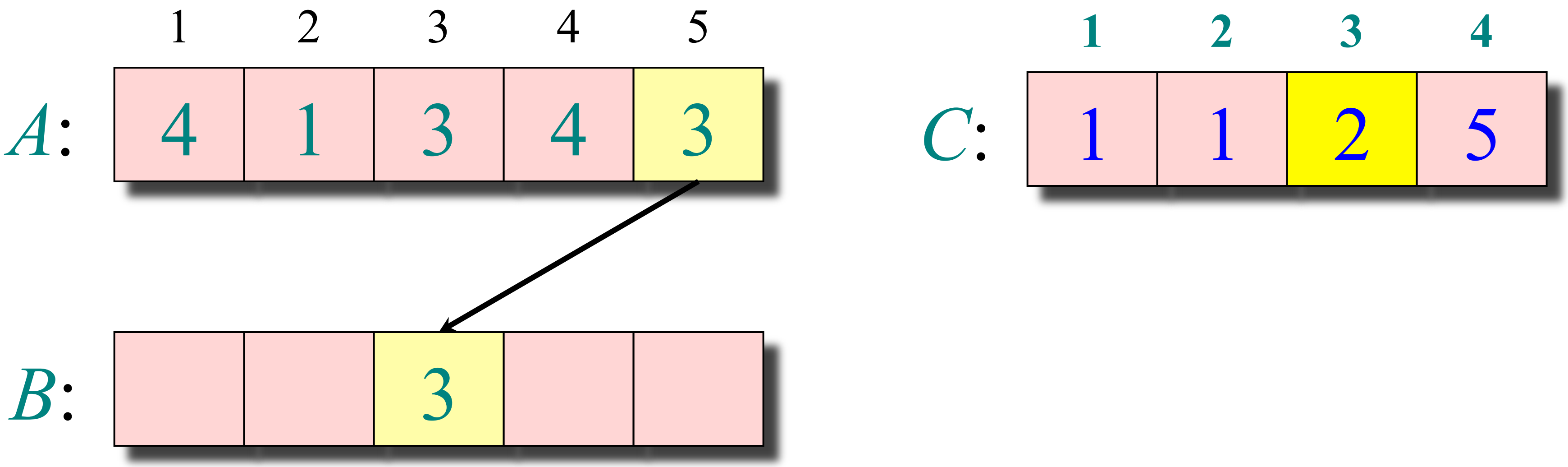


	1	2	3	4
C:	1	1	3	5

Used-up one 3; update counter.

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

Loop 4: permute elements of A



```
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4: permute elements of A

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	1	2	5

B:			3		

```
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4: permute elements of A

	1	2	3	4	5
A:	4	1	3	4	3

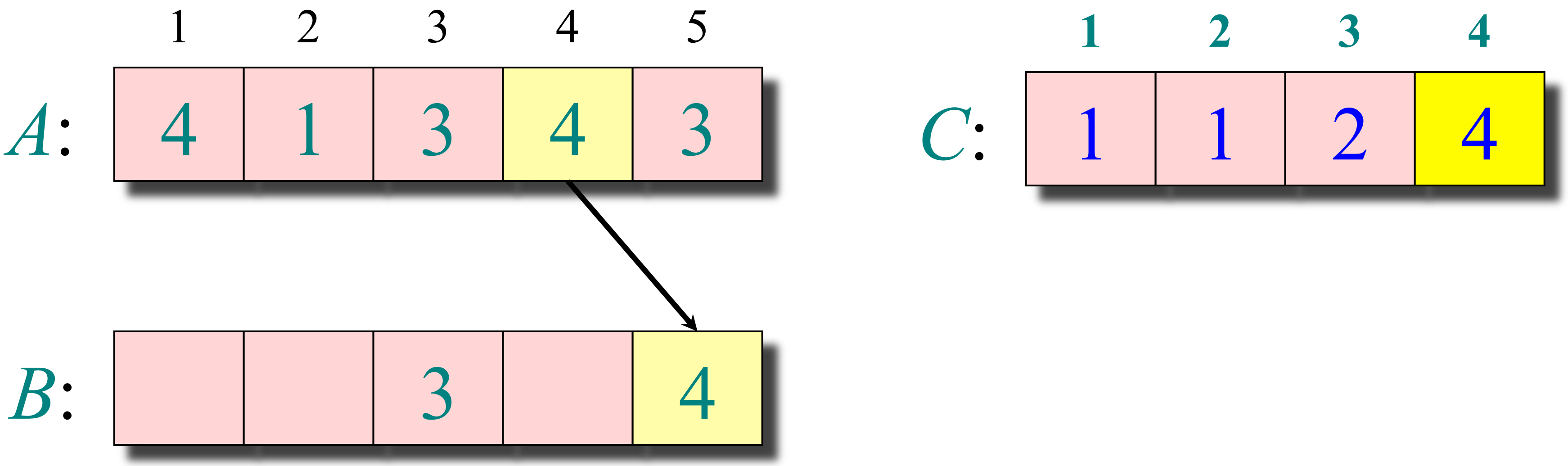
	1	**2**	**3**	**4**
C:	1	1	2	5

B:			3		

There are exactly 5 elements ≤A[4], so where should I place A[4]?

```
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4: permute elements of A



	1	2	3	4	5
A:	4	1	3	4	3

B:			3		4

	1	2	3	4
C:	1	1	2	4

```
for $j \leftarrow n$ downto 1
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4: permute elements of A

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	1	2	4

B:			3		4

```
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4: permute elements of A

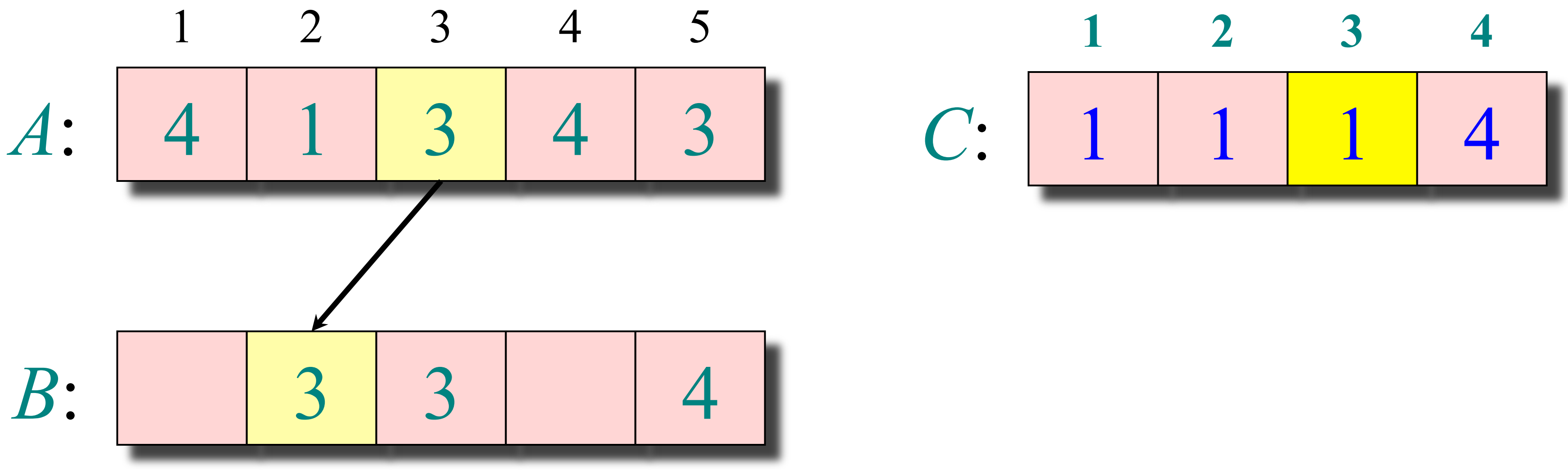
	1	2	3	4	5
A:	4	1	3	4	3

	1	**2**	**3**	**4**
C:	1	1	2	4

B:			3		4

```
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4: permute elements of A



	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	1	1	4

B:		3	3		4

```
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4: permute elements of A

	1	2	3	4	5
A:	4	1	3	4	3
B:		3	3		4

	1	2	3	4
C:	1	1	1	4

```
for $j \leftarrow n$ downto 1
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$
```

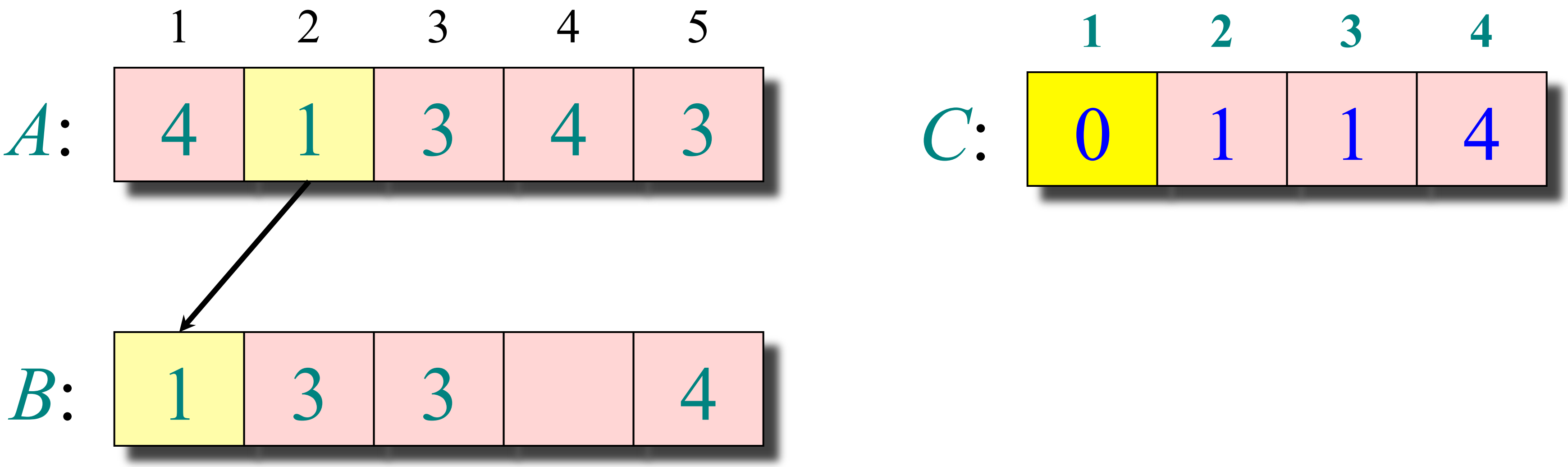
Loop 4: permute elements of A

	1	2	3	4	5
A:	4	1	3	4	3
B:		3	3		4

	1	2	3	4
C:	1	1	1	4

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

Loop 4: permute elements of A



	1	2	3	4	5
A:	4	1	3	4	3
B:	1	3	3		4

	1	2	3	4
C:	0	1	1	4

```
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4: permute elements of A

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	0	1	1	4

B:	1	3	3		4

```
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4: permute elements of A

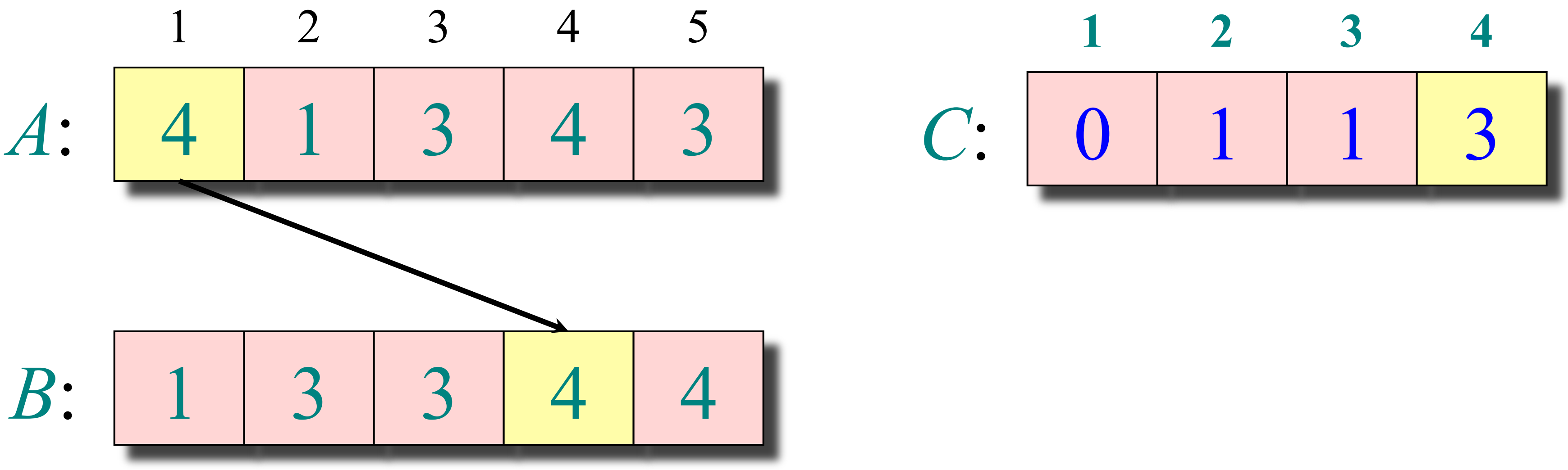
	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	0	1	1	4

B:	1	3	3		4

```
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4: permute elements of A



	1	2	3	4	5
A:	4	1	3	4	3
B:	1	3	3	4	4

	1	2	3	4
C:	0	1	1	3

```
for $j \leftarrow n$ downto 1
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Analysis

```
$\Theta(k)$       for $i \leftarrow 1$ to $k$
                      do $C[i] \leftarrow 0$
$\Theta(n)$       for $j \leftarrow 1$ to $n$
                      do $C[A[j]] \leftarrow C[A[j]] + 1$
$\Theta(k)$       for $i \leftarrow 2$ to $k$
                      do $C[i] \leftarrow C[i] + C[i-1]$
$\Theta(n)$       for $j \leftarrow n$ downto $1$
                      do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
                         $C[A[j]] \leftarrow C[A[j]] - 1$
```

$\Theta(n + k)$

Running time

If $k = O(n)$, then counting sort takes $\Theta(n)$ time.

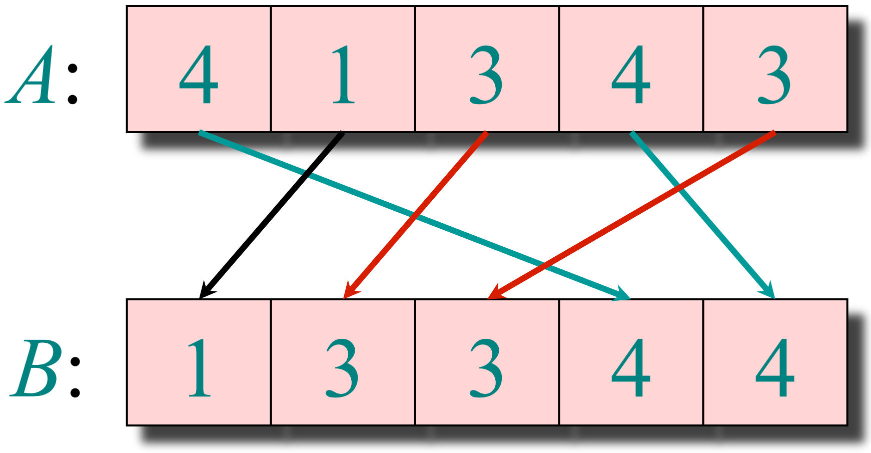
- But, sorting takes $\Omega(n \lg n)$ time!
- Where's the fallacy?

Answer:

- ***Comparison sorting*** takes $\Omega(n \lg n)$ time.
- Counting sort is not a ***comparison sort***.
- In fact, not a single comparison between elements occurs!

Stable sorting

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



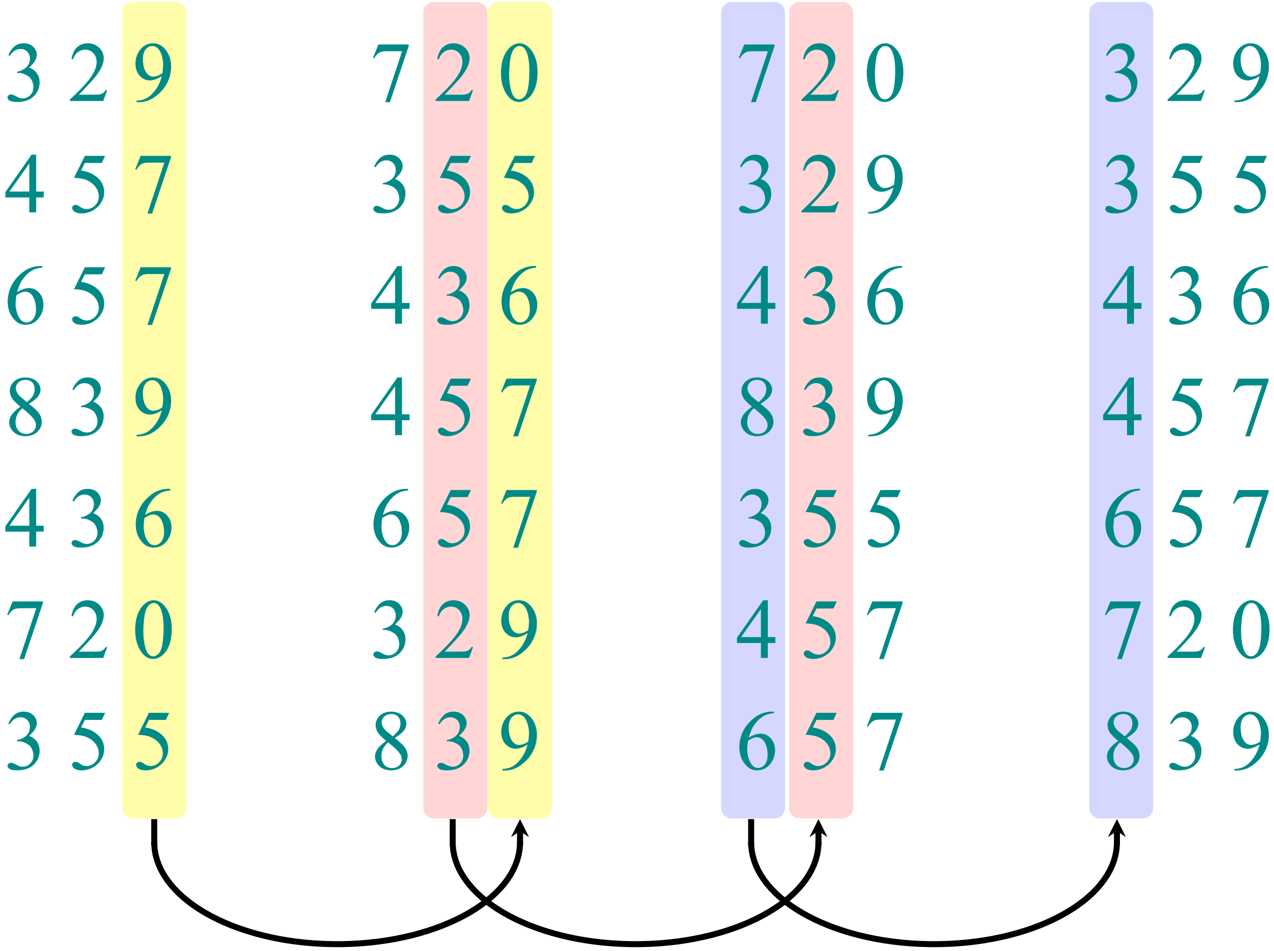
A: 4 1 3 4 3

B: 1 3 3 4 4

Radix sort

- ***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.

Operation of radix sort

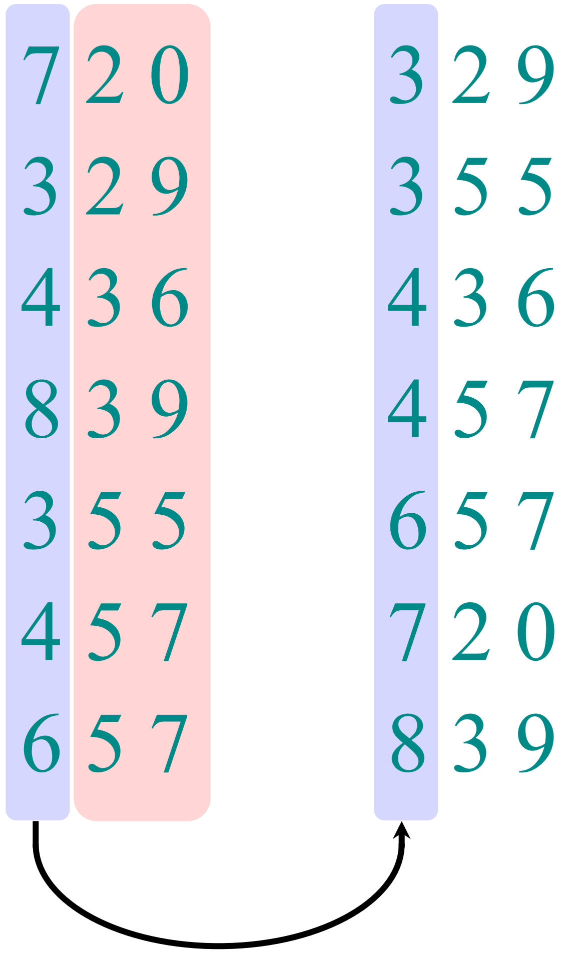


329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

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

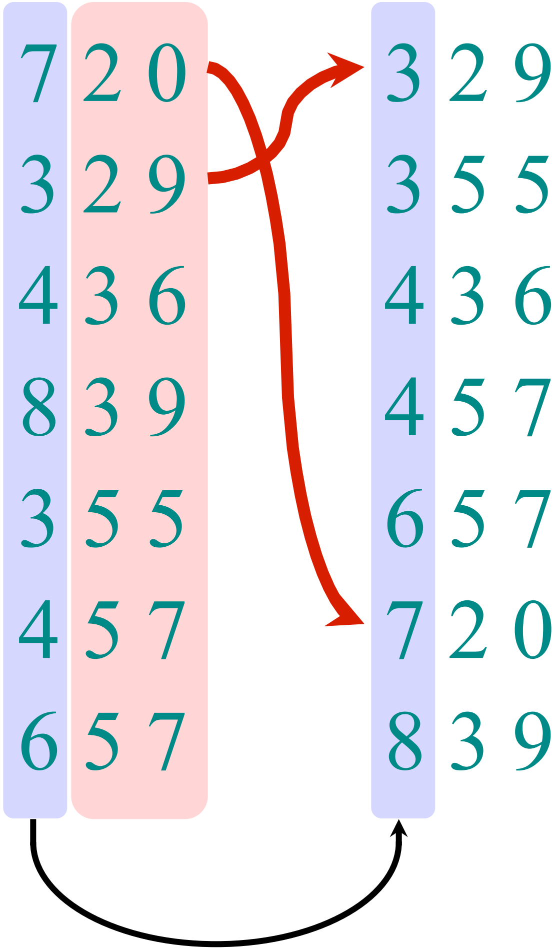


7 2 0	3 2 9
3 2 9	3 5 5
4 3 6	4 3 6
8 3 9	4 5 7
3 5 5	6 5 7
4 5 7	7 2 0
6 5 7	8 3 9

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.

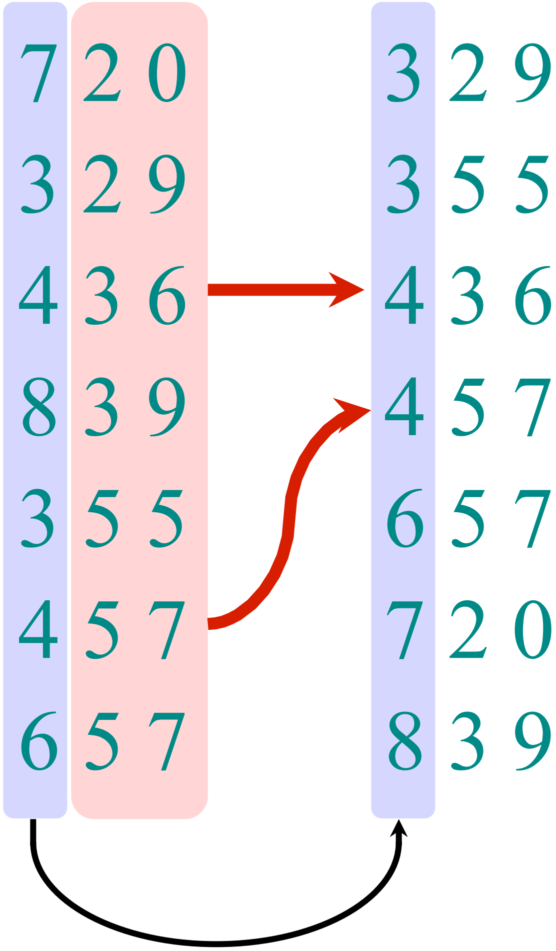


7 2 0	3 2 9
3 2 9	3 5 5
4 3 6	4 3 6
8 3 9	4 5 7
3 5 5	6 5 7
4 5 7	7 2 0
6 5 7	8 3 9

Correctness of radix sort

Induction on digit position

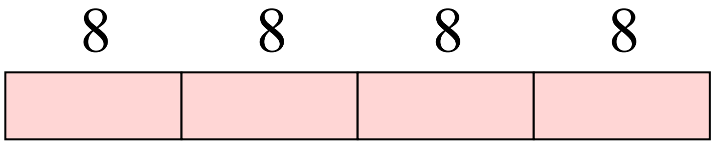
- Assume that the numbers are sorted by their low-order $t - 1$ digits.
- Sort on digit t
  - Two numbers that differ in digit t are correctly sorted.
  - Two numbers equal in digit t are put in the same order as the input ⇒ correct order.



Runtime 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: 32-bit word



- If each b-bit word is broken into r-bit pieces, each pass of counting sort takes $\Theta(n + 2^r)$ time.
- Setting r=log n gives $\Theta(n)$ time per pass, or

$\Theta(n\ b/\log n)$ total