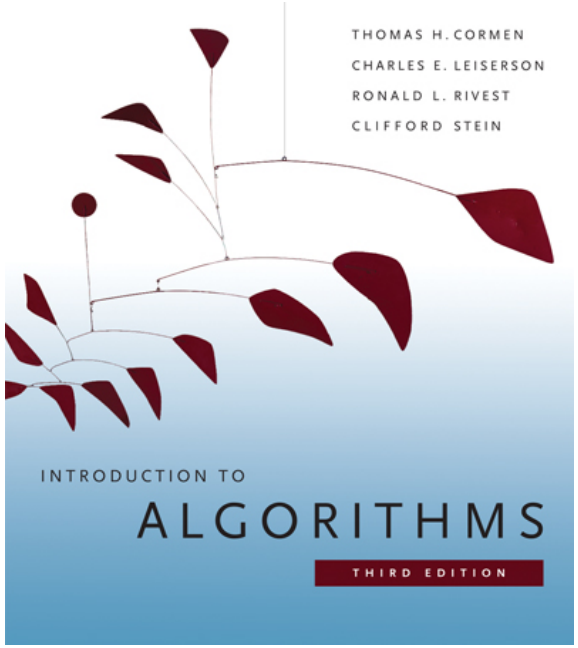
6.006- *Introduction to Algorithms*



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.

So the elements could be numbers, water-samples compared on the basis of their concentration in chloride, etc.

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

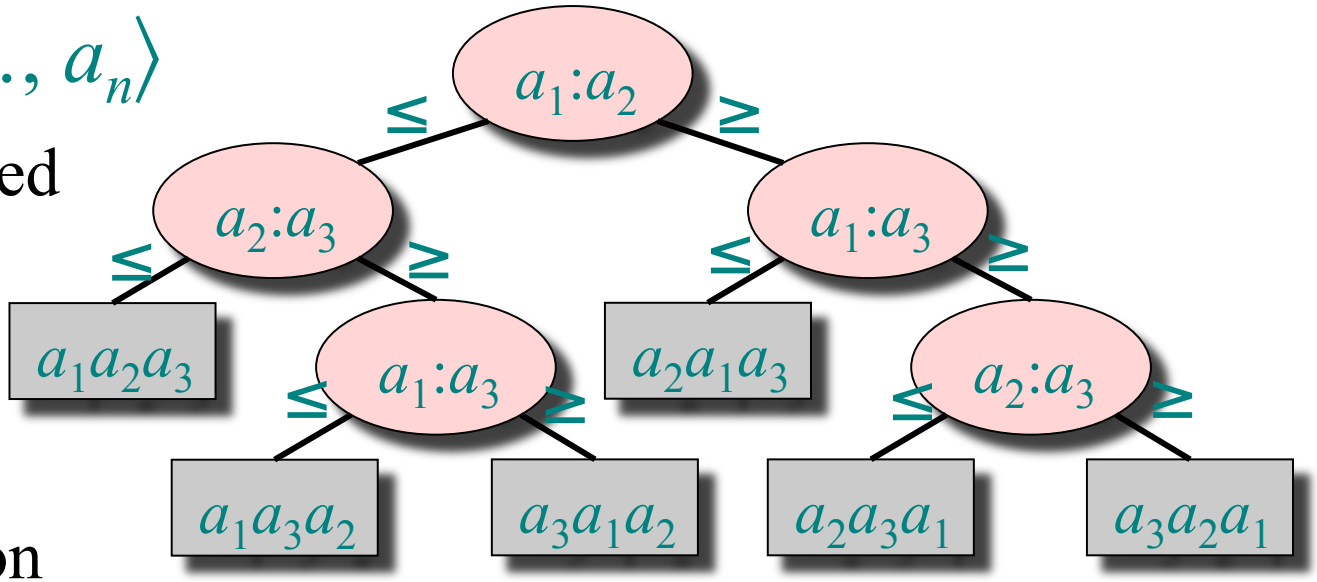
Is* $\mathbf{O(n \log n)}$ *the best we can do?

Decision trees can help us answer this question.

Decision-tree

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

- Nodes are suggested comparisons:
  $a_i:a_j$ means compare a_i to a_j.

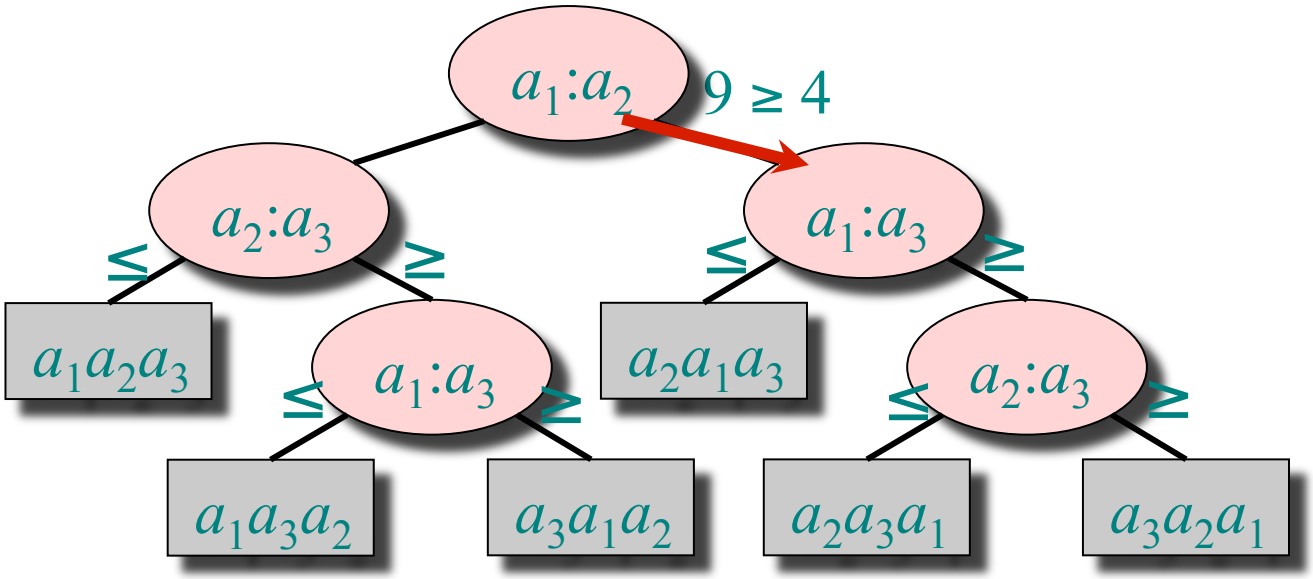


- Branching direction depends on outcome of comparisons.

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

Decision-tree example

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

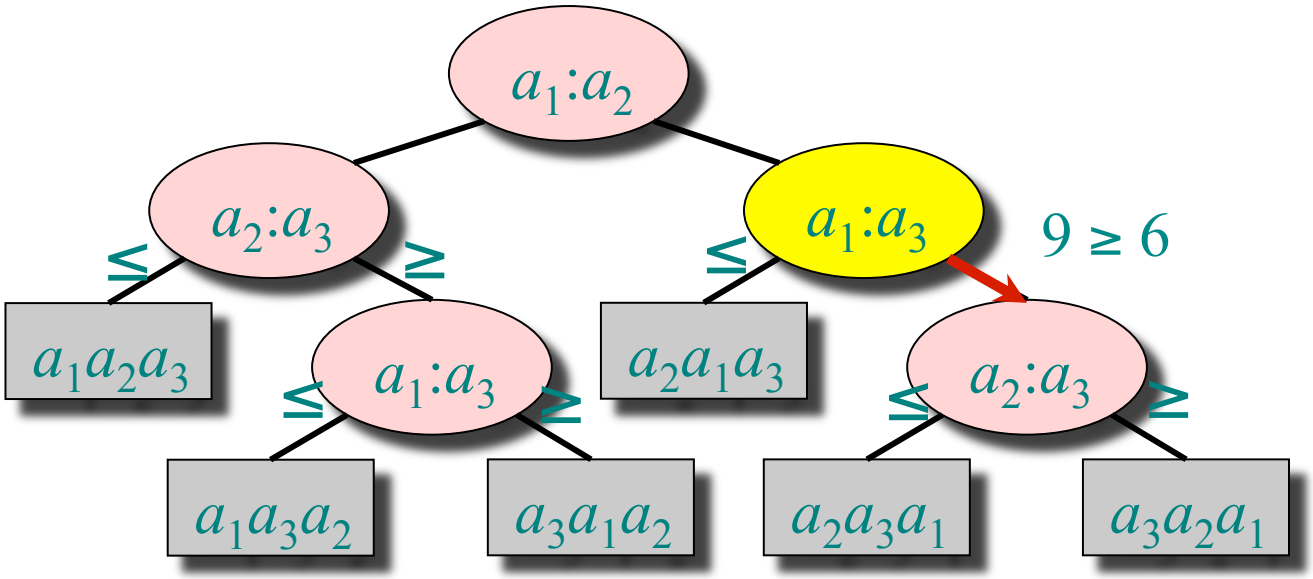


Each internal node is labeled $a_i{:}a_j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.
- Each leaf contains a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ to indicate that the ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$ was found.

Decision-tree example

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

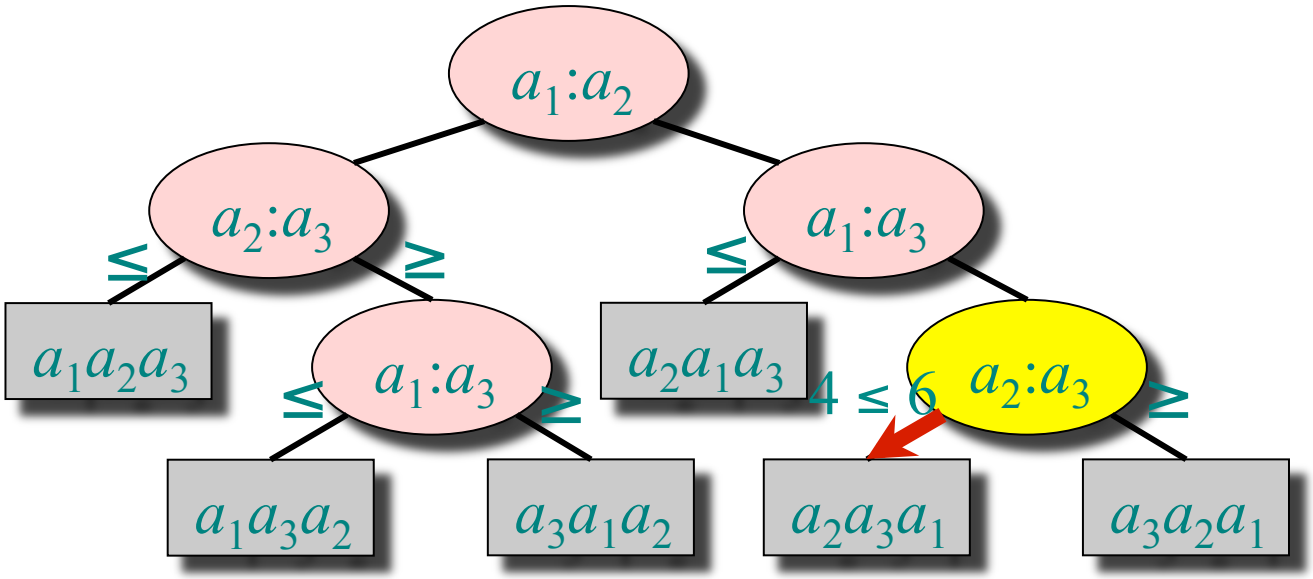


Each internal node is labeled $a_i{:}a_j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.
- Each leaf contains a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ to indicate that the ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$ was found.

Decision-tree example

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

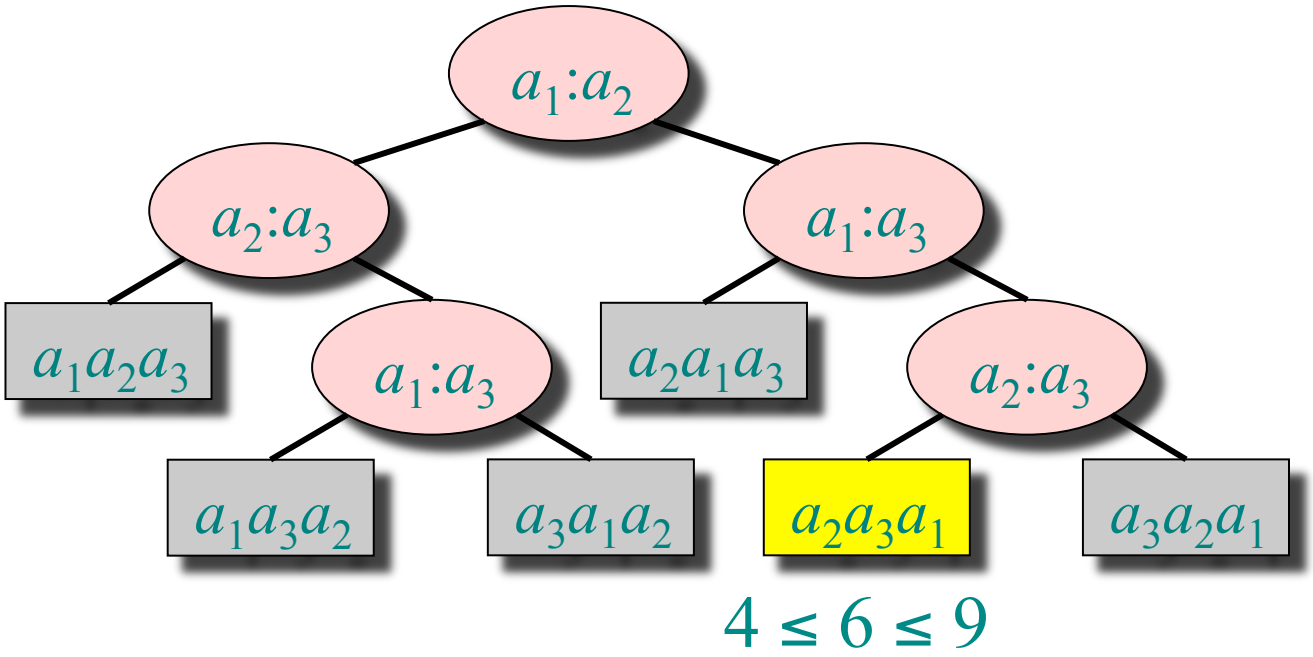


Each internal node is labeled $a_i{:}a_j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \le a_j$.
- The right subtree shows subsequent comparisons if $a_i \ge a_j$.
- Each leaf contains a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ to indicate that the ordering $a_{\pi(1)} \le a_{\pi(2)} \le \cdots \le a_{\pi(n)}$ was found.

Decision-tree example

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



Each internal node is labeled $a_i{:}a_j$ 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 $a_i \ge a_j$.
- Each leaf contains a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ to indicate that the ordering $a_{\pi(1)} \le a_{\pi(2)} \le \cdots \le a_{\pi(n)}$ was found.

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 for n elements must have height $\Omega(n \log n)$.

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.
- For it to be able to sort it must be that:

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

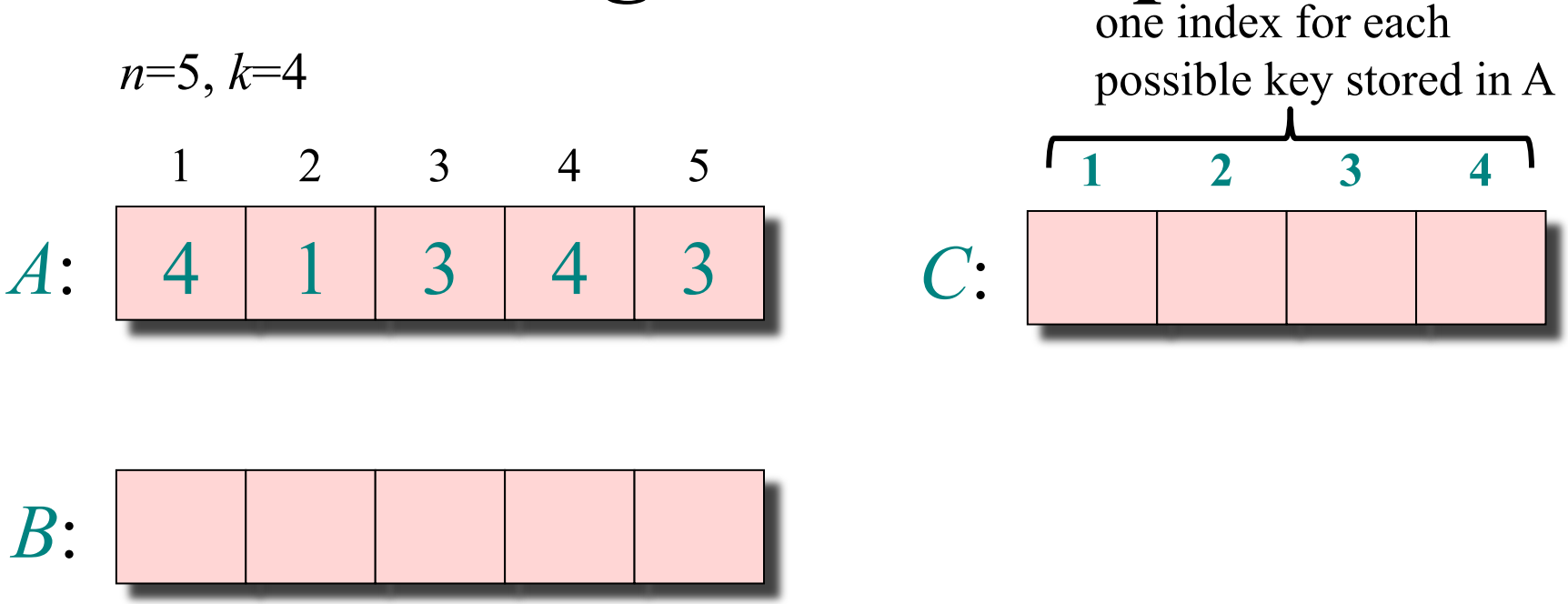
Sorting in linear time

Counting sort: No comparisons between elements.

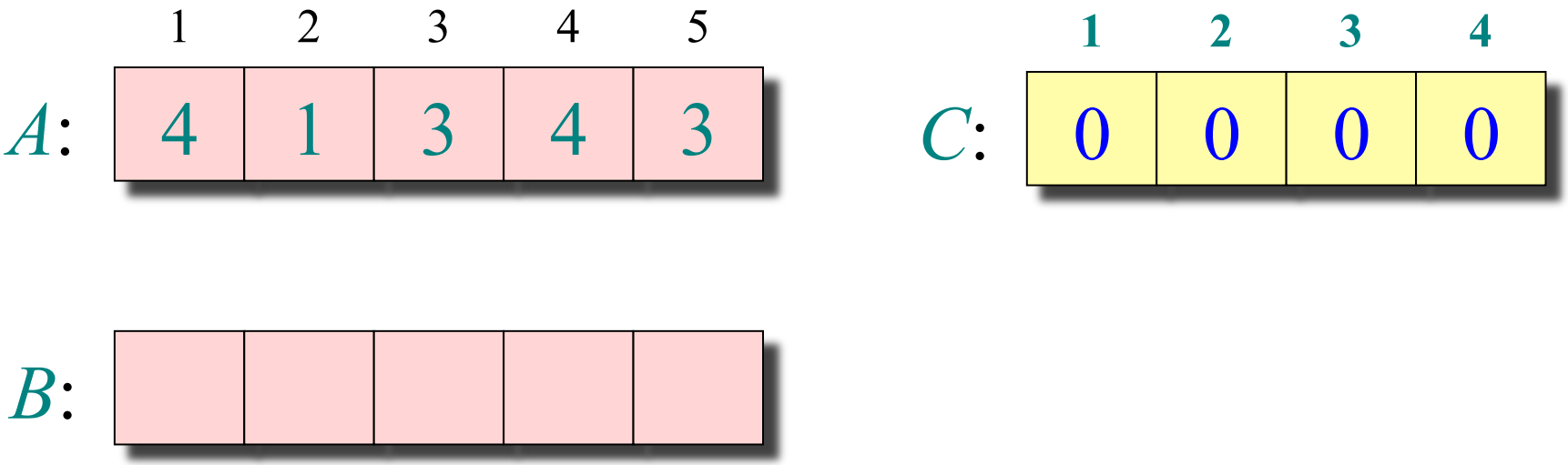
- ***Input***: $A[1 \,..\, n]$, where $A[j] \in \{1, 2, \ldots, k\}$.
- ***Output***: $B[1 \,..\, n]$, a sorted permutation of A
- ***Auxiliary storage***: $C[1 \,..\, k]$.

Counting-sort example

n=5, k=4



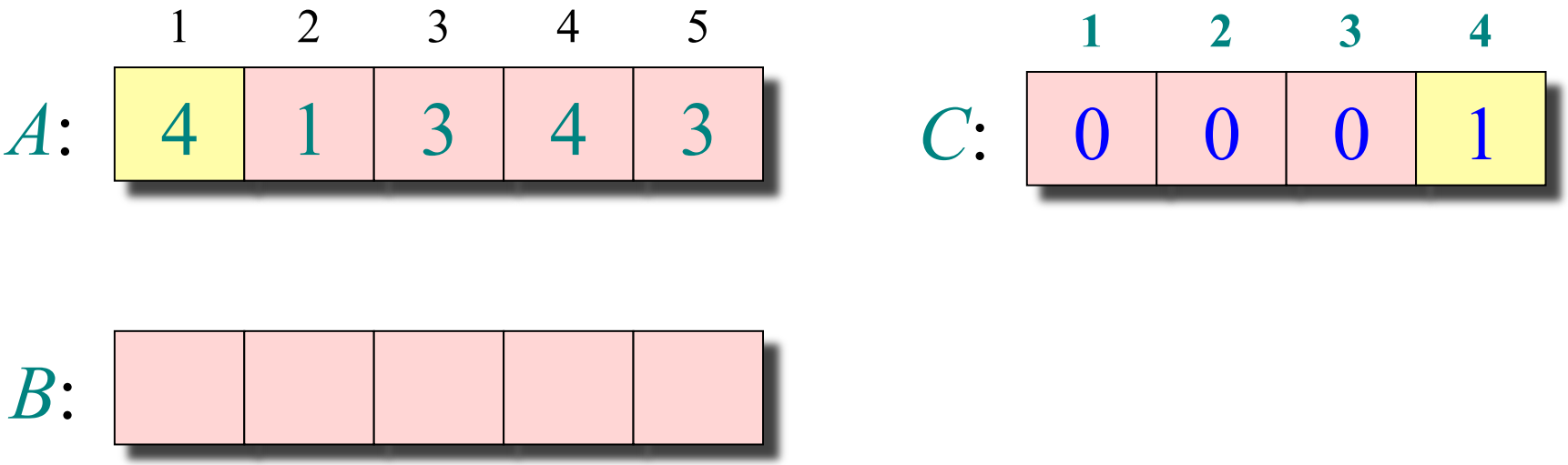
Loop 1: initialization



for $i \leftarrow 1$ **to** k

do $C[i] \leftarrow 0$

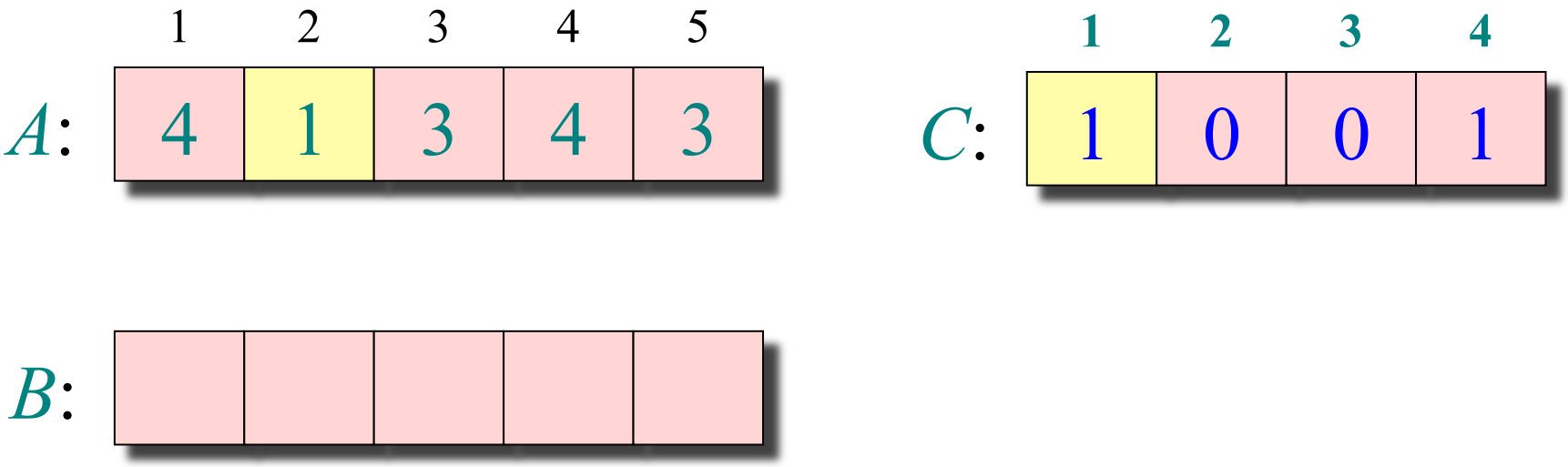
Loop 2: count frequencies



for $j \leftarrow 1$ **to** n

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

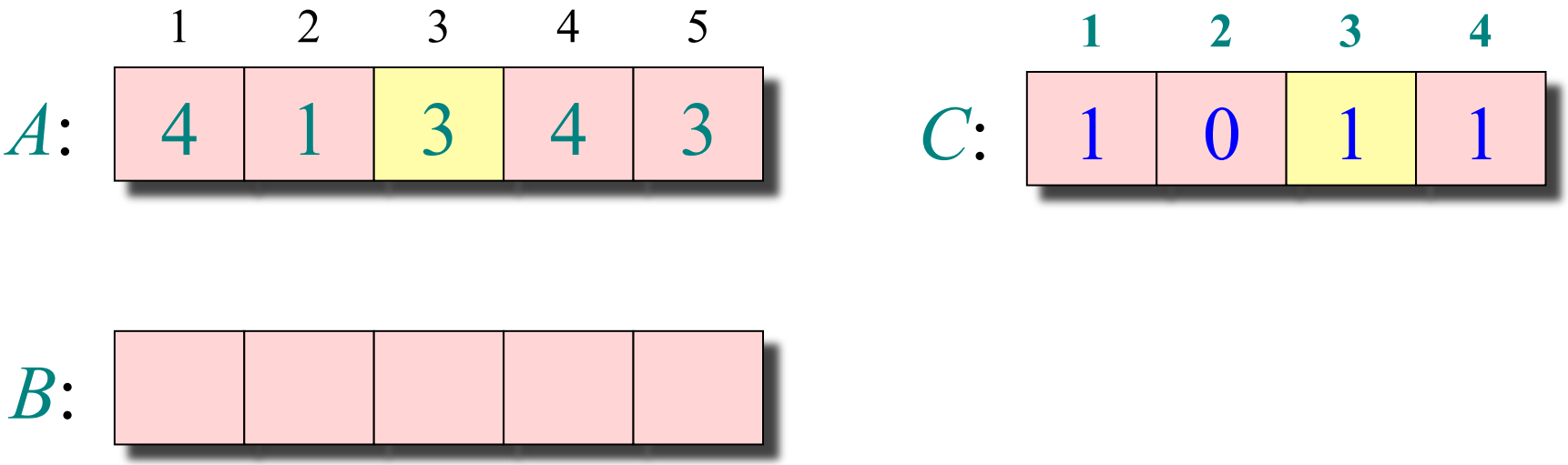
Loop 2: count frequencies



for $j \leftarrow 1$ **to** n

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

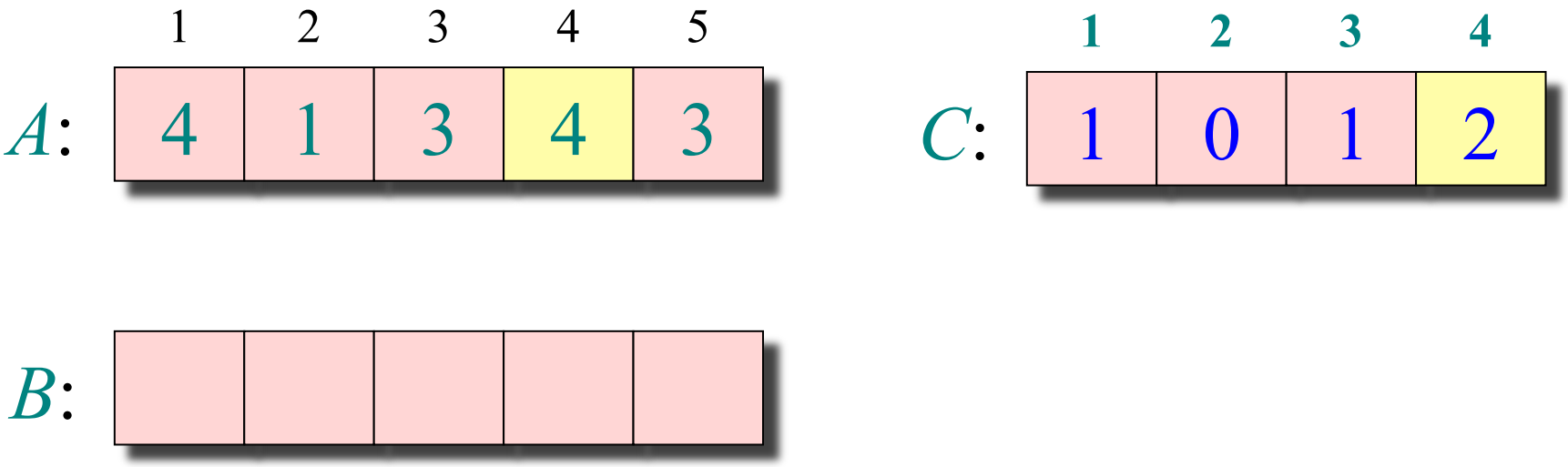
Loop 2: count frequencies



for $j \leftarrow 1$ **to** n

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

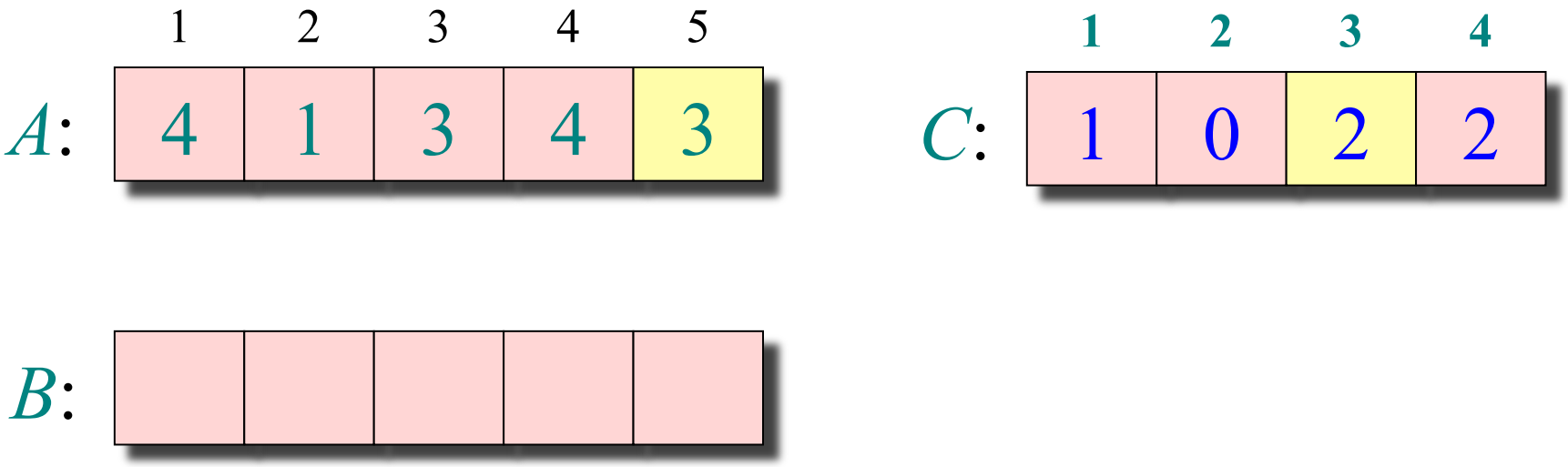
Loop 2: count frequencies



for $j \leftarrow 1$ **to** n

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

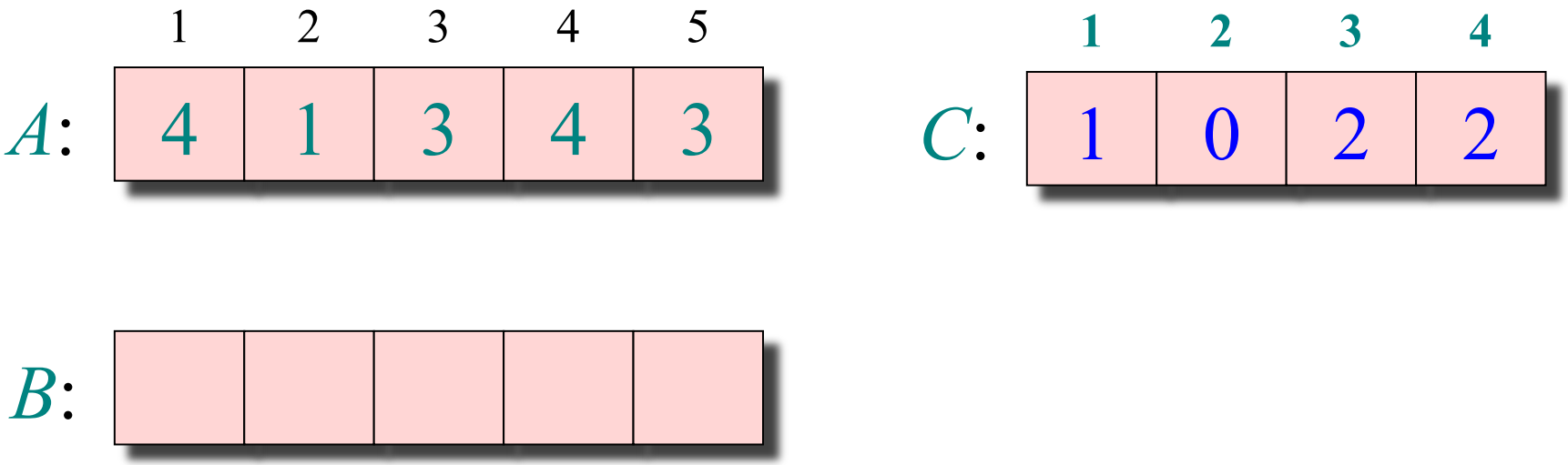
Loop 2: count frequencies



for $j \leftarrow 1$ **to** n

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

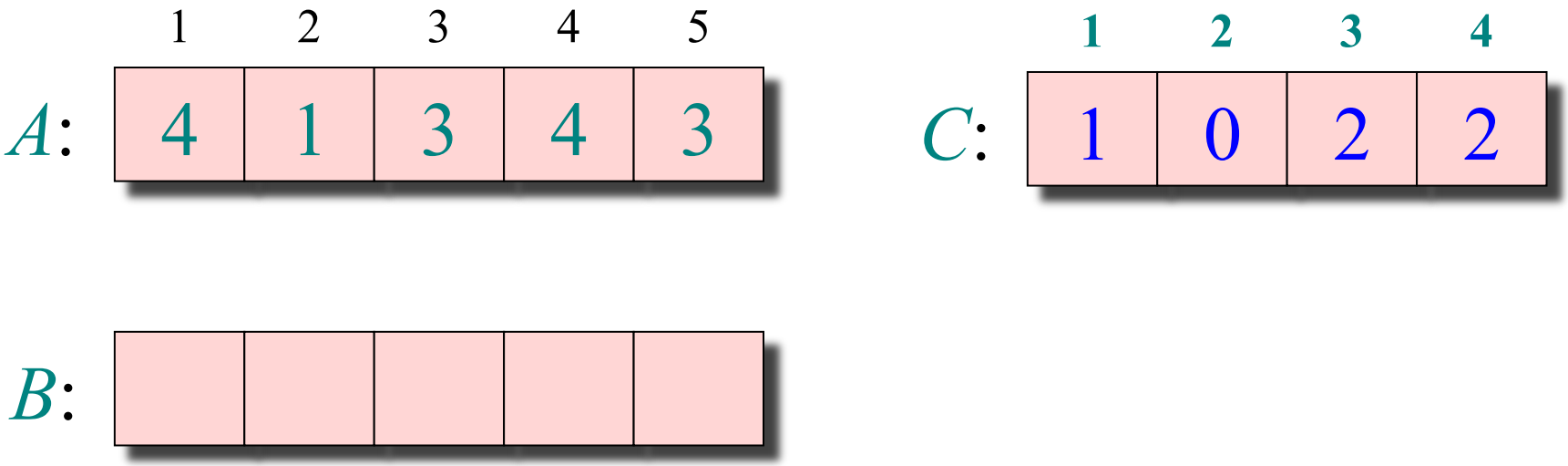
Loop 2: count frequencies



for $j \leftarrow 1$ **to** n

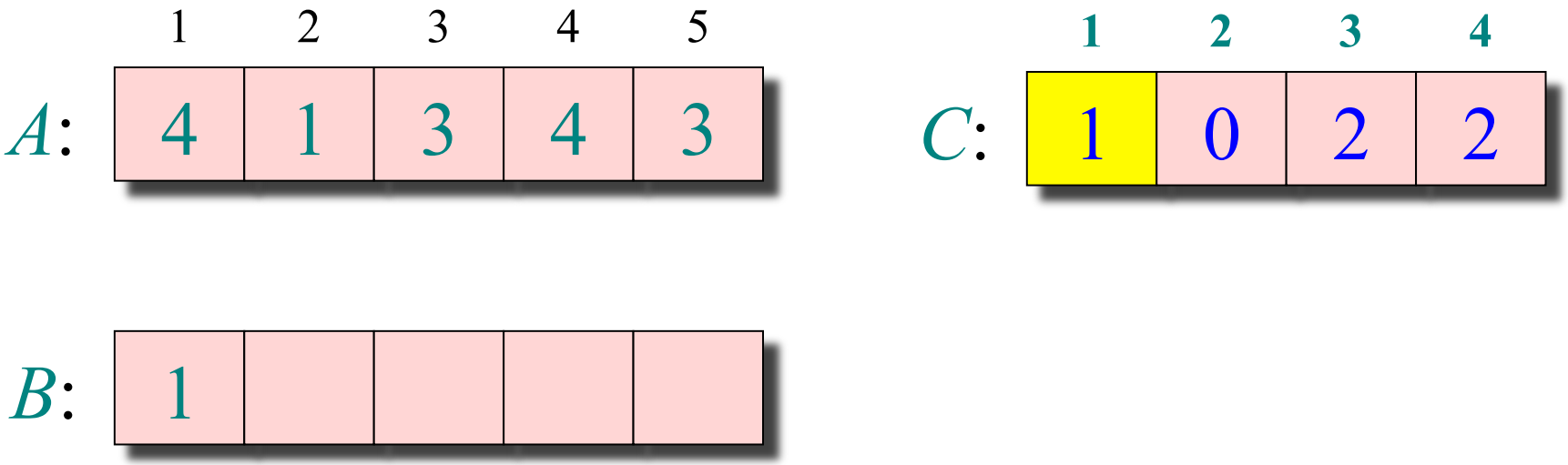
do $C[A[j]] \leftarrow C[A[j]] + 1$ ▷ $C[i] = |\{\text{key} = i\}|$

[A parenthesis: a quick finish

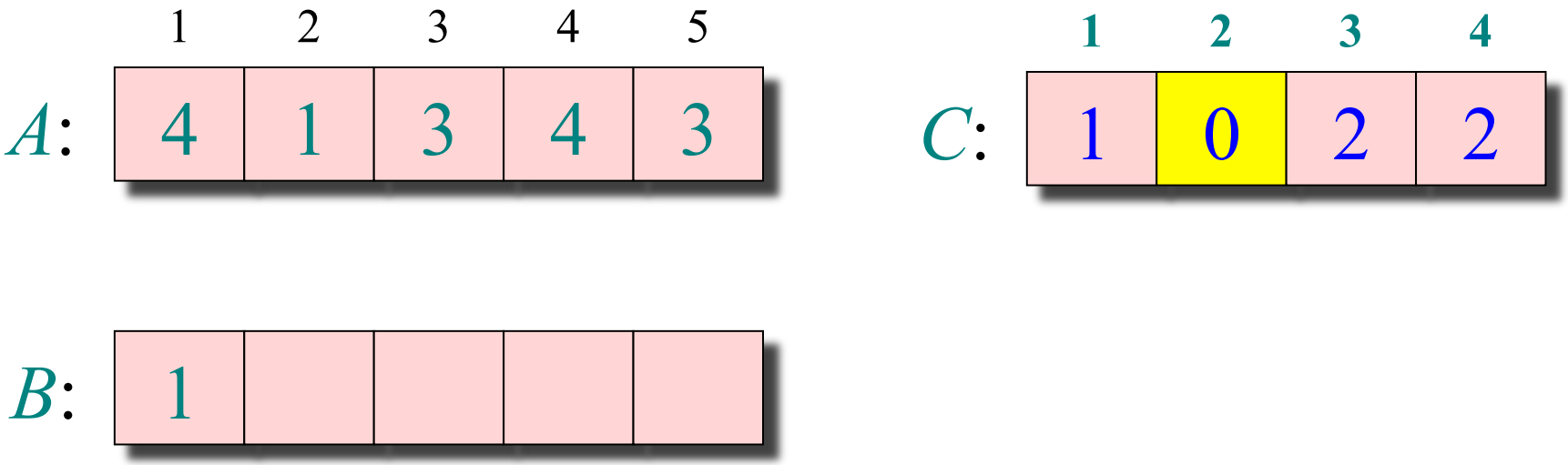


Walk through frequency array and place the appropriate number of each key in output array…

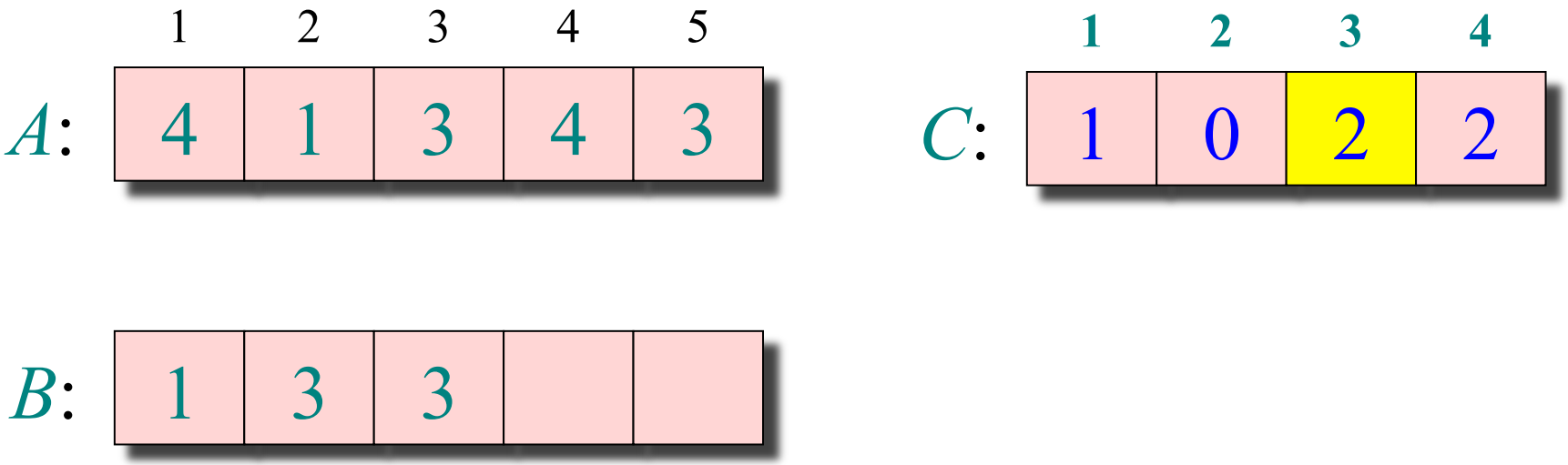
A parenthesis: a quick finish



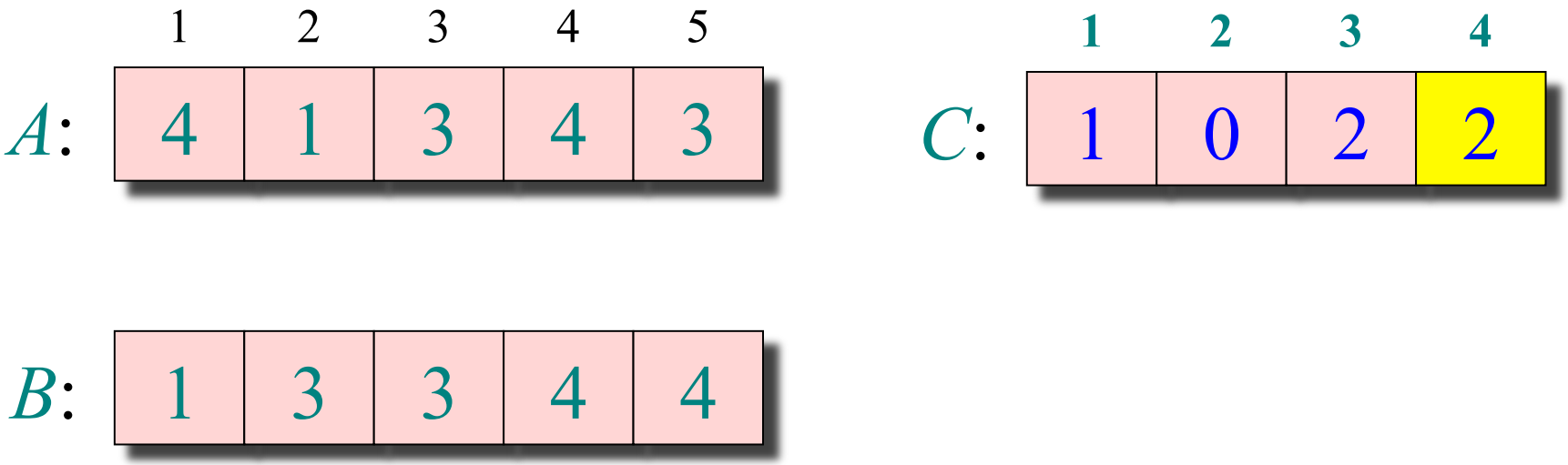
A parenthesis: a quick finish



A parenthesis: a quick finish



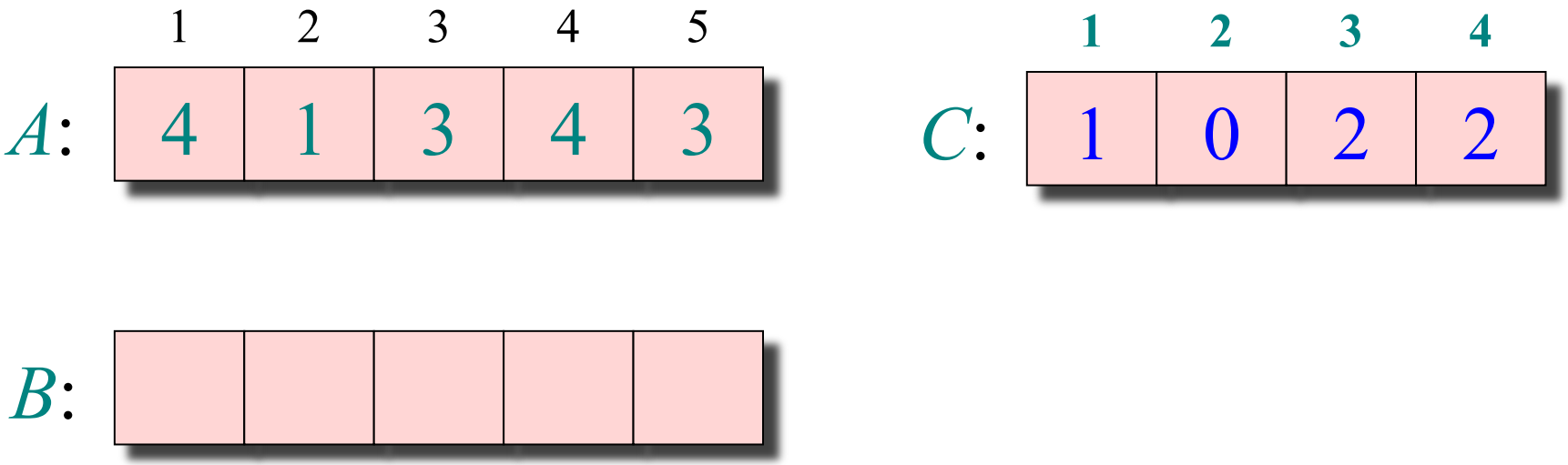
A parenthesis: a quick finish



B is sorted!

but it is not "stably sorted"…]

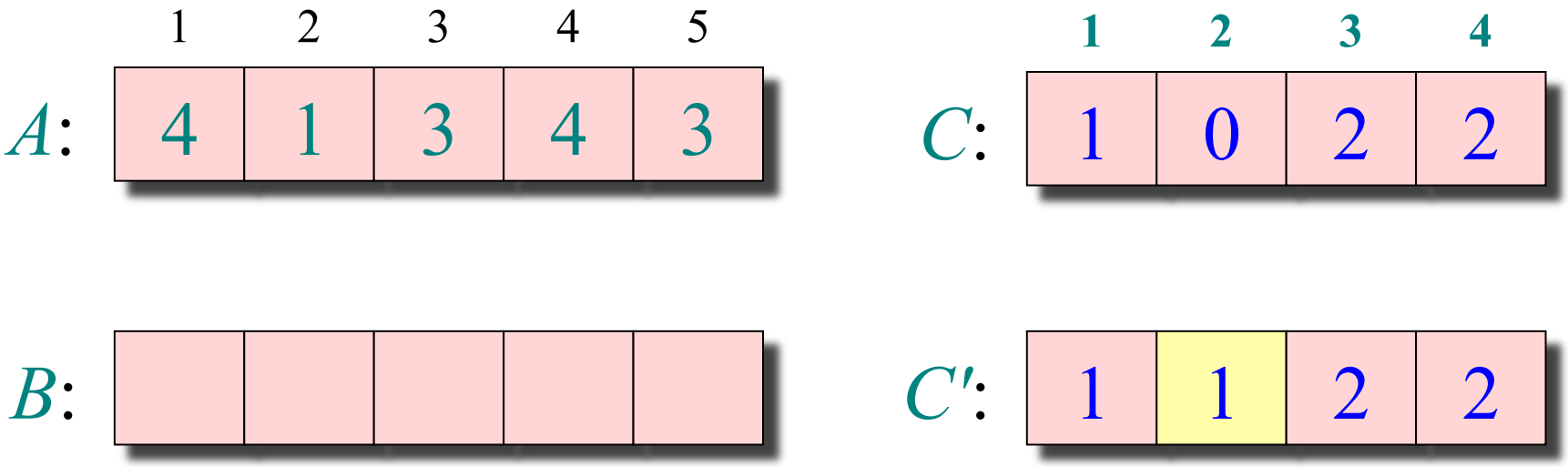
Loop 3: from frequencies to cumulative frequencies…



for $i \leftarrow 2$ **to** k

do $C[i] \leftarrow C[i] + C[i-1]$ ▷ $C[i] = |\{\text{key} \le i\}|$

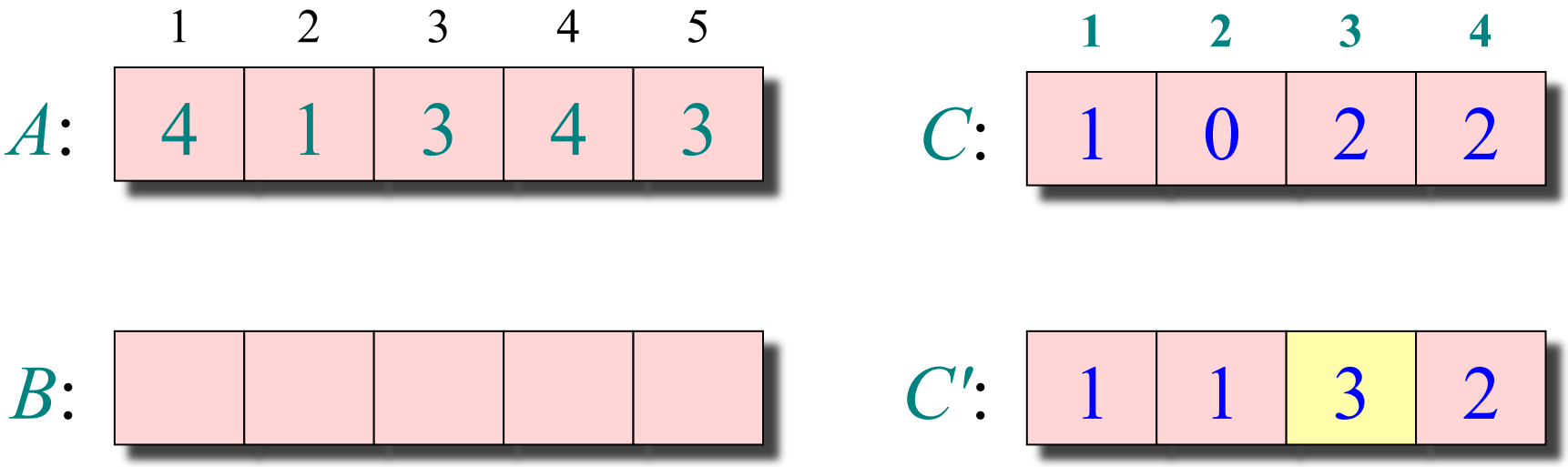
Loop 3: from frequencies to cumulative frequencies…



for $i \leftarrow 2$ **to** k

  do $C[i] \leftarrow C[i] + C[i-1]$ ▷ $C[i] = |\{\text{key} \leq i\}|$

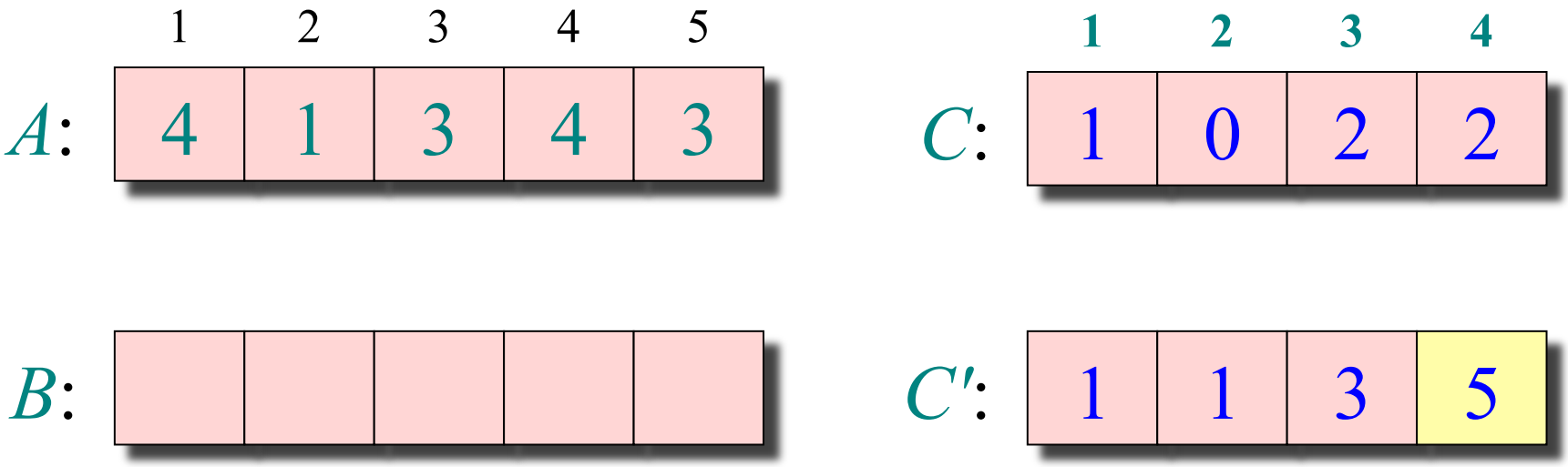
Loop 3: from frequencies to cumulative frequencies…



for $i \leftarrow 2$ **to** k

  do $C[i] \leftarrow C[i] + C[i-1]$ ▷ $C[i] = |\{\text{key} \leq i\}|$

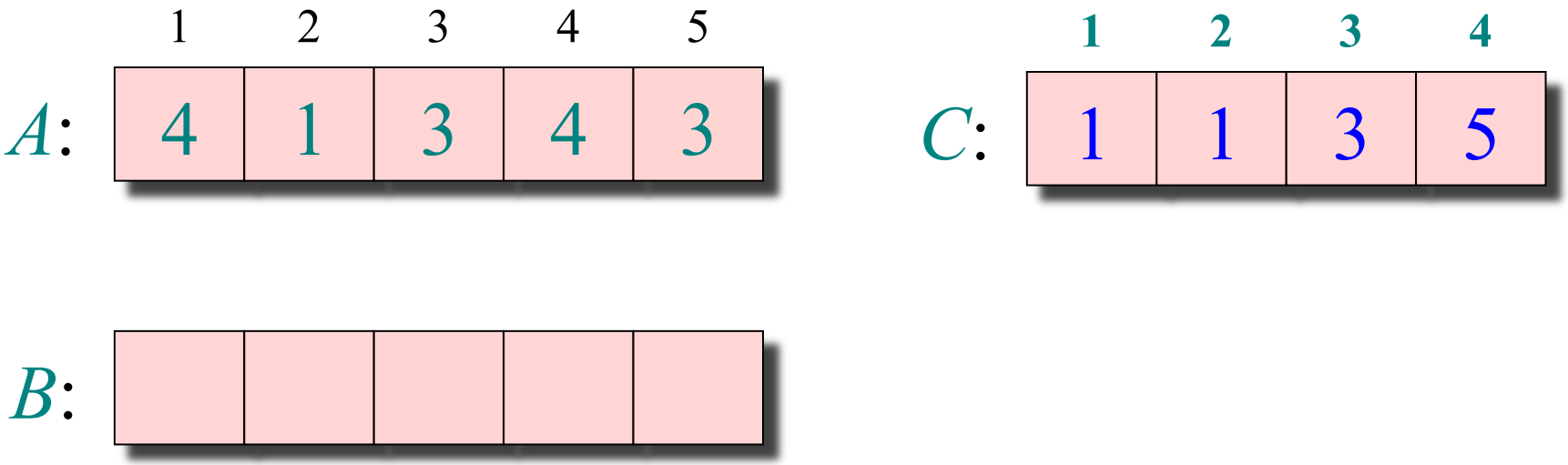
Loop 3: from frequencies to cumulative frequencies…



for $i \leftarrow 2$ **to** k

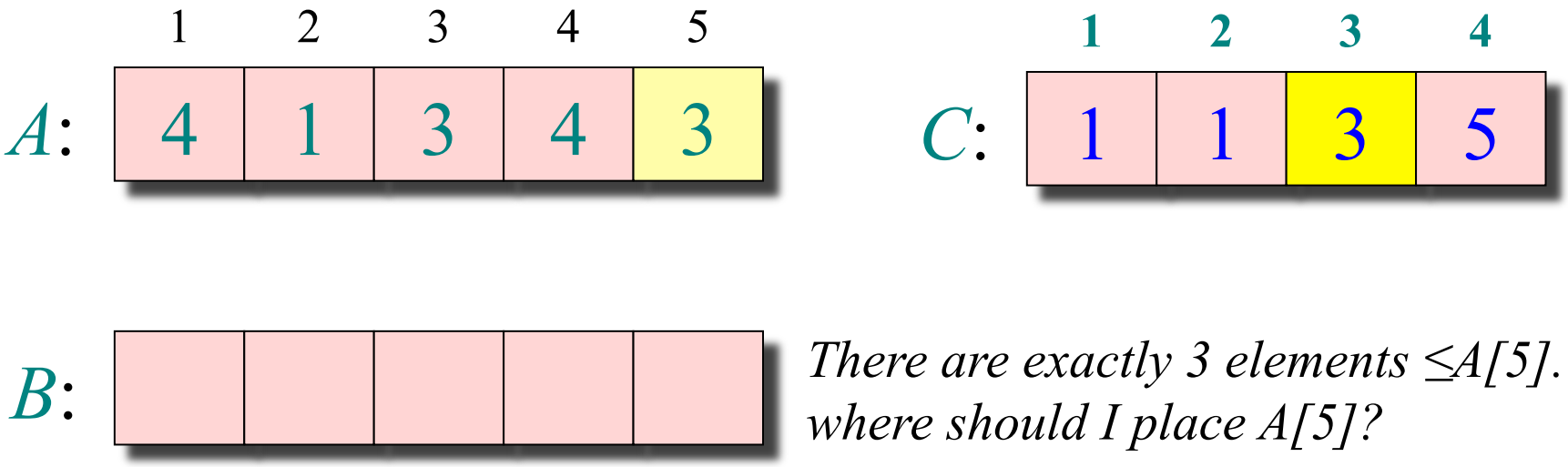
  do $C[i] \leftarrow C[i] + C[i-1]$ ▷ $C[i] = |\{\text{key} \leq i\}|$

Loop 4: permute elements of A



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

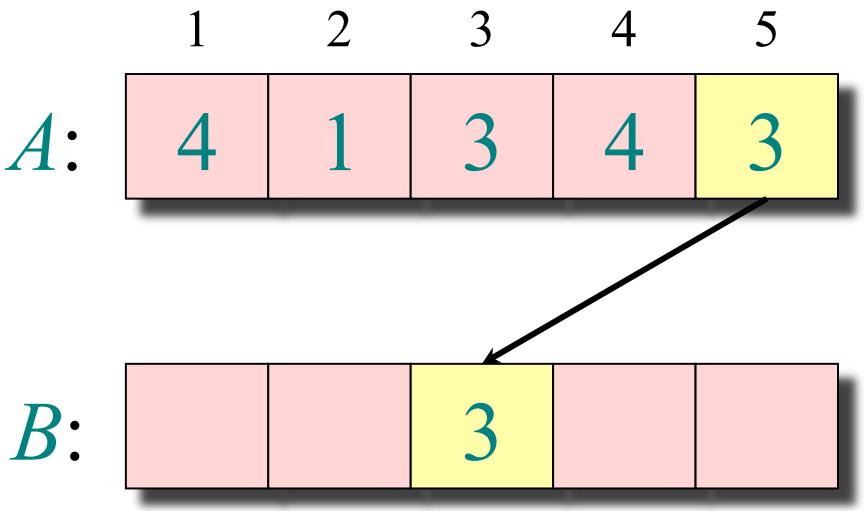
Loop 4: permute elements of A

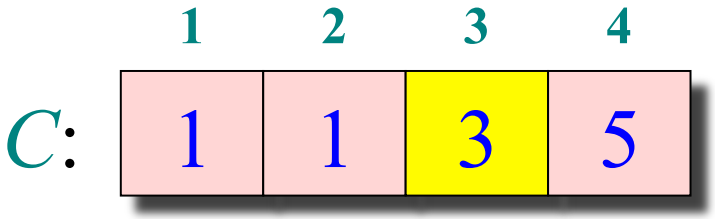


There are exactly 3 elements ≤A[5]. So where should I place A[5]?

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

Loop 4: permute elements of A

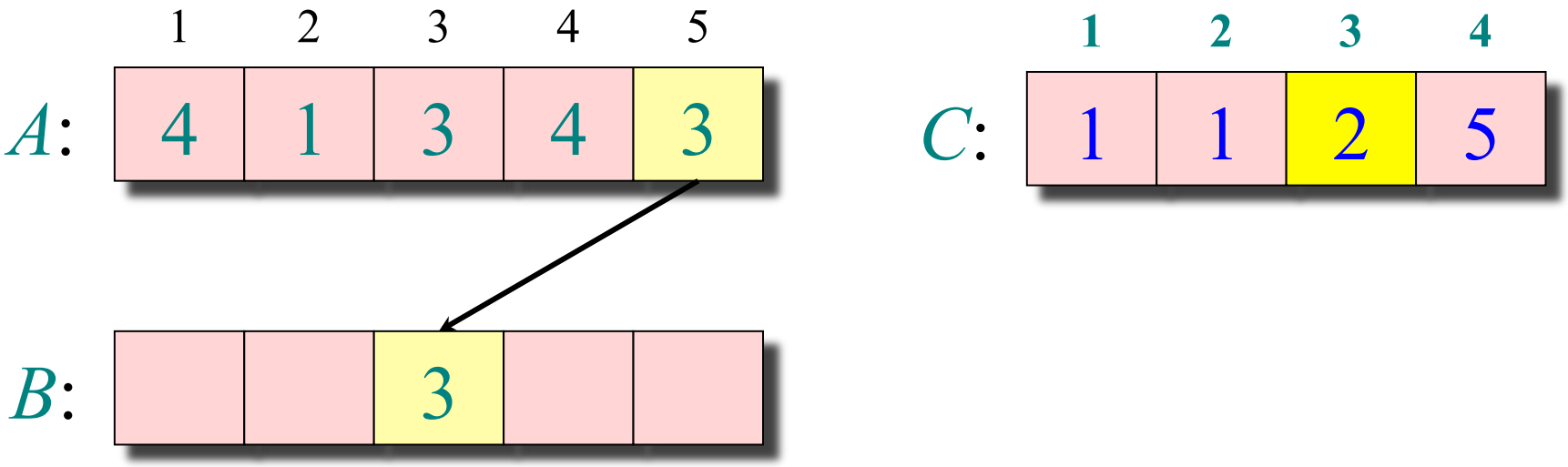




Used-up one 3; update counter in C for the next 3 that shows up...

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

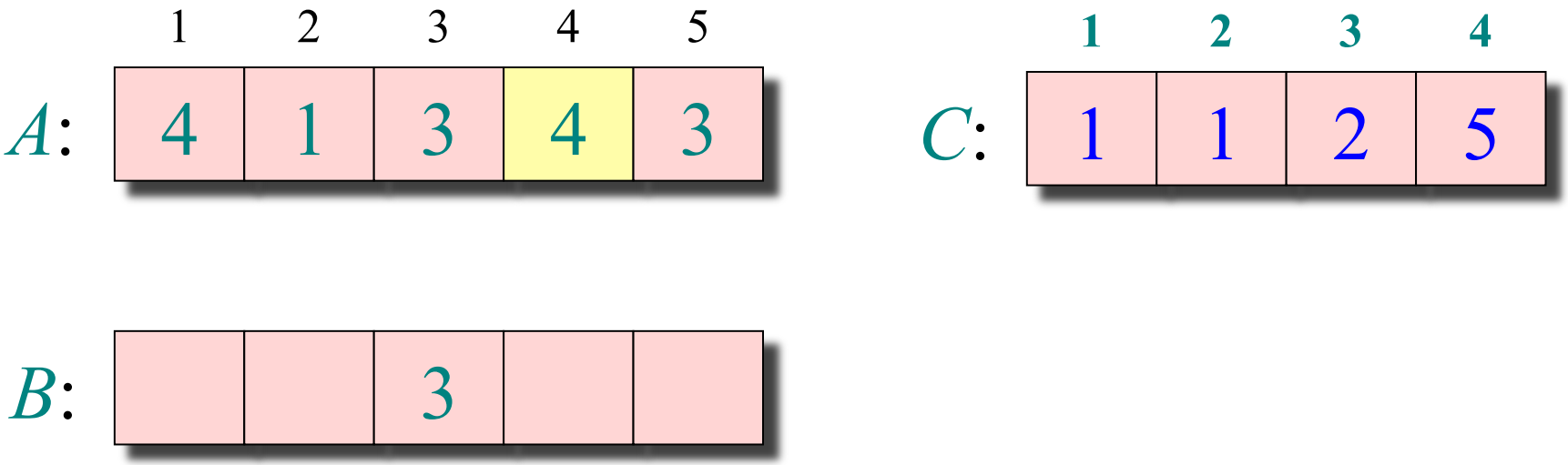


for $j \leftarrow n$ **downto** 1

do $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A

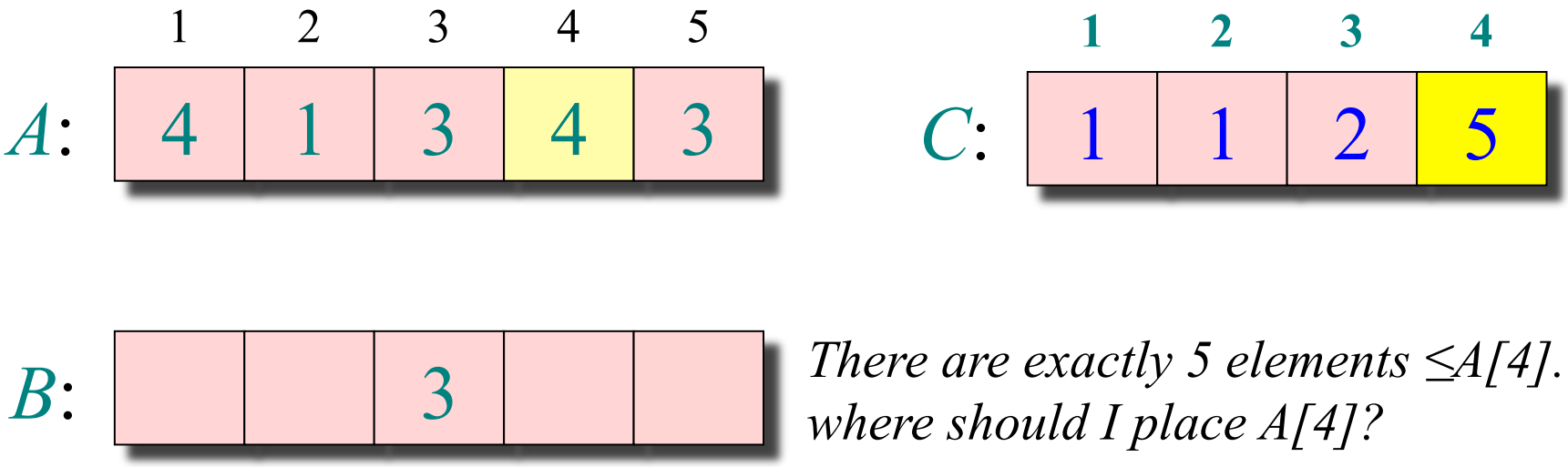


for $j \leftarrow n$ **downto** 1

do $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A



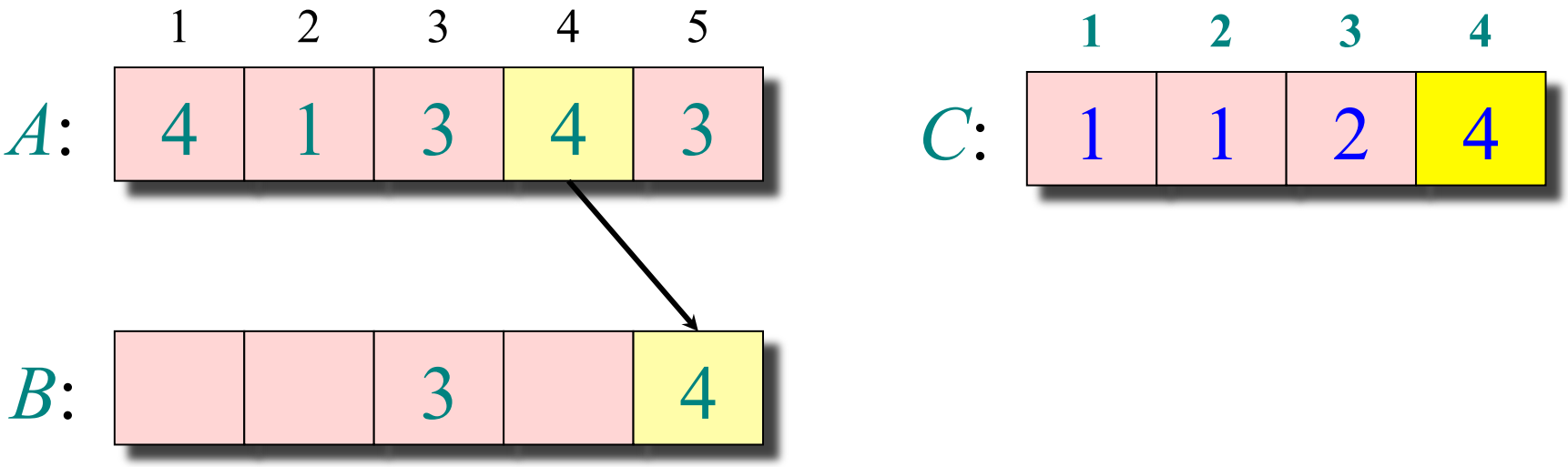
There are exactly 5 elements ≤A[4]. So where should I place A[4]?

for $j \leftarrow n$ **downto** 1

do $B[C[A[j]]] \leftarrow A[j]$

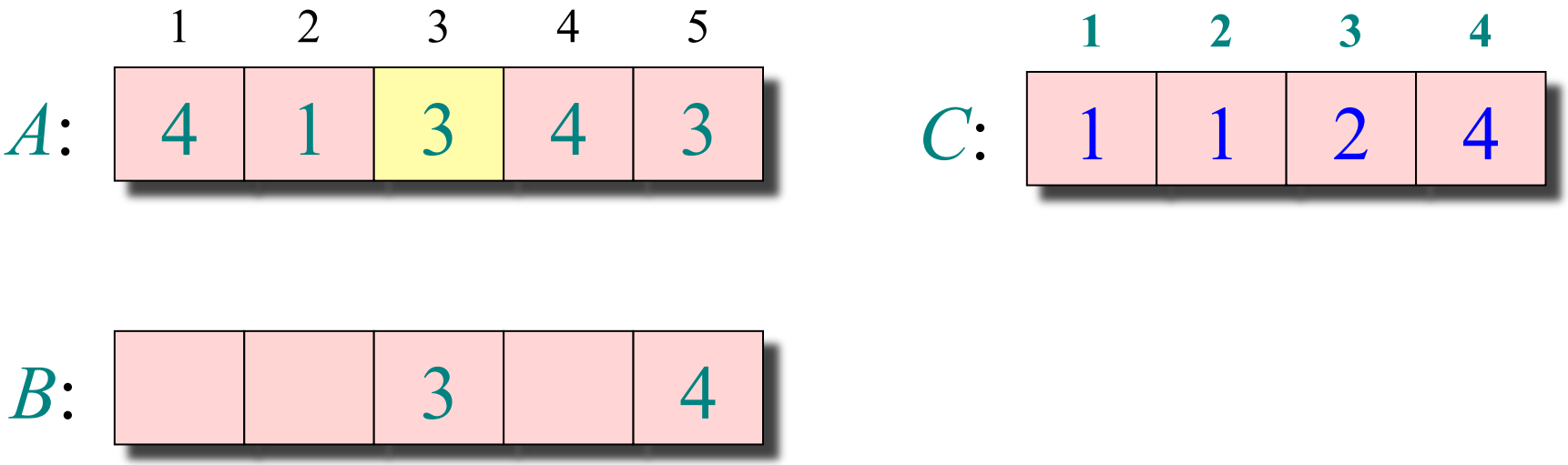
$C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A



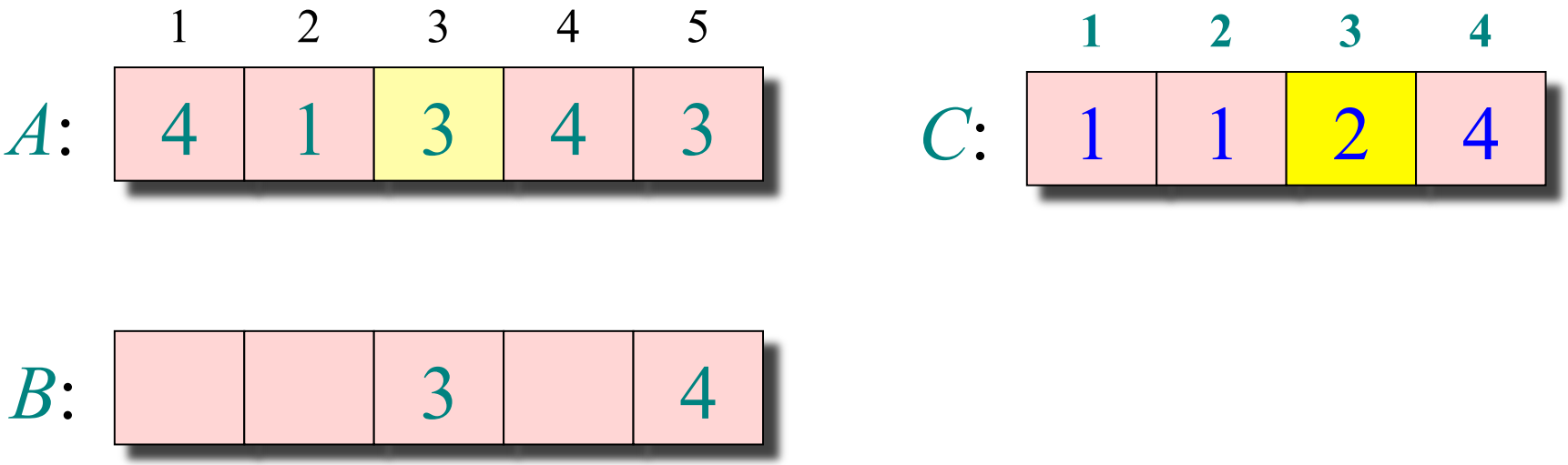
for $j \leftarrow n$ **downto** 1
    do $B[C[A[j]]] \leftarrow A[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A



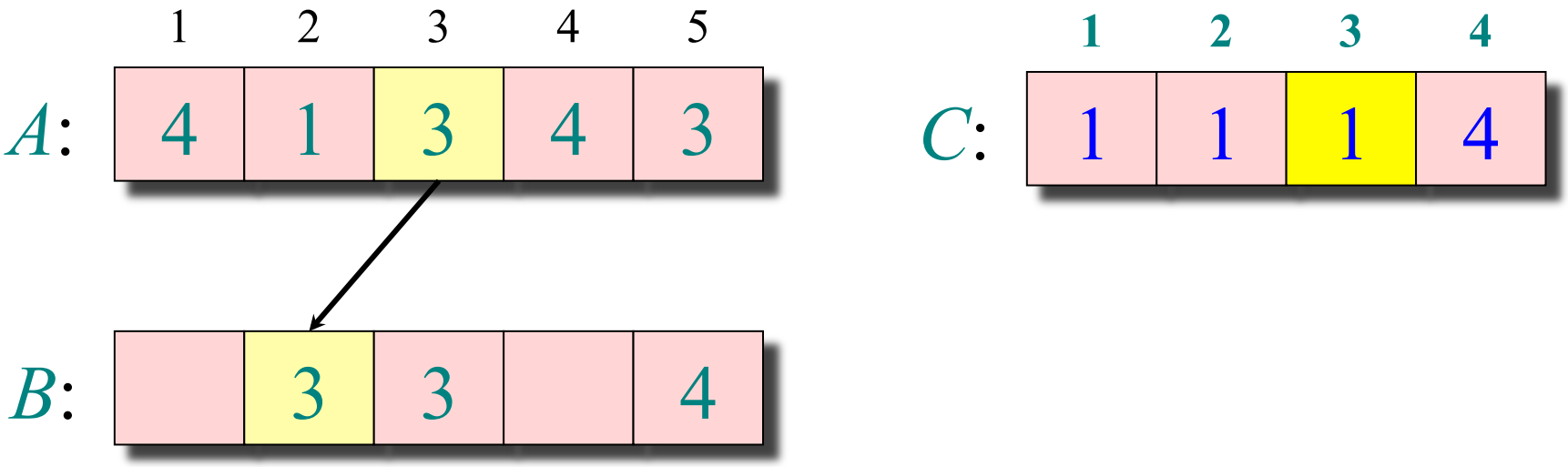
for $j \leftarrow n$ **downto** 1
    do $B[C[A[j]]] \leftarrow A[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A



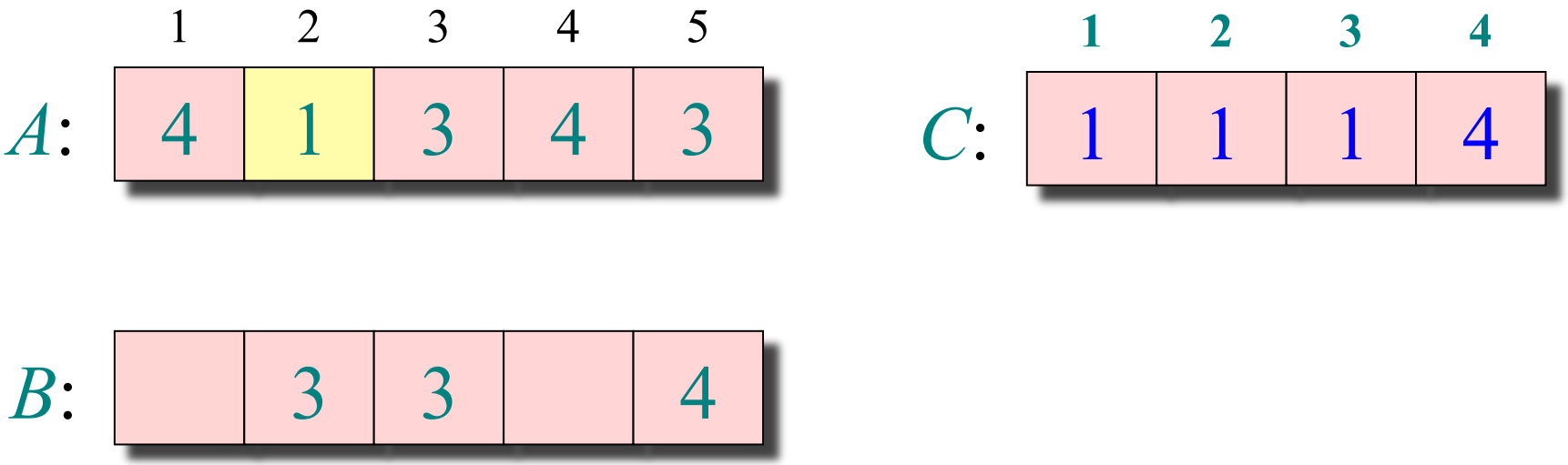
for $j \leftarrow n$ **downto** 1
    do $B[C[A[j]]] \leftarrow A[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A



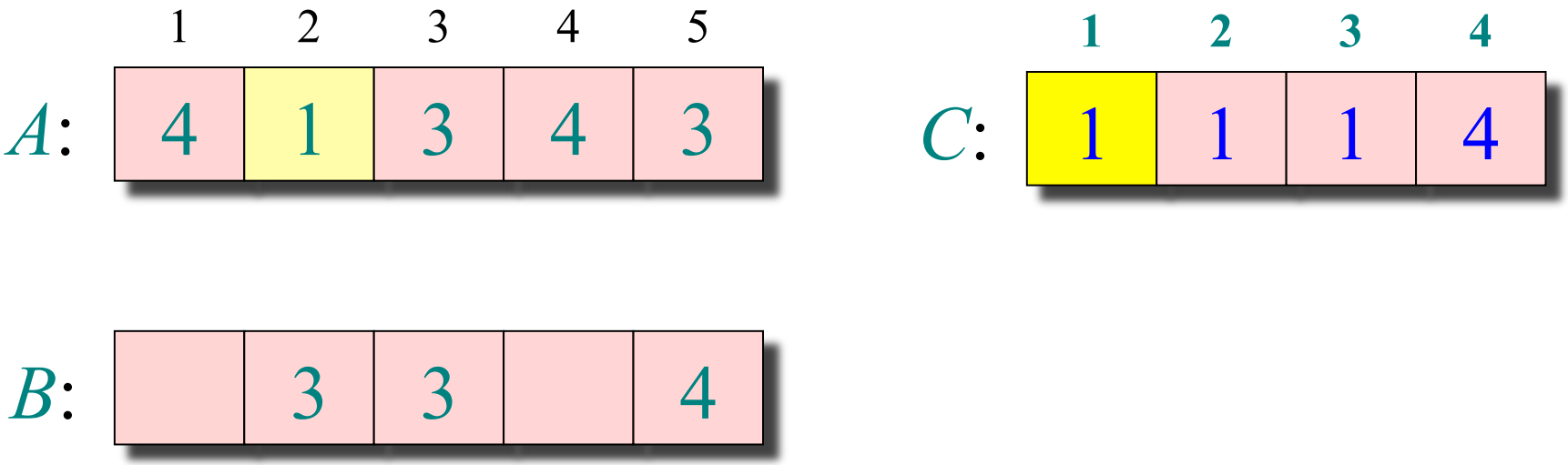
for $j \leftarrow n$ **downto** 1
    do $B[C[A[j]]] \leftarrow A[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A



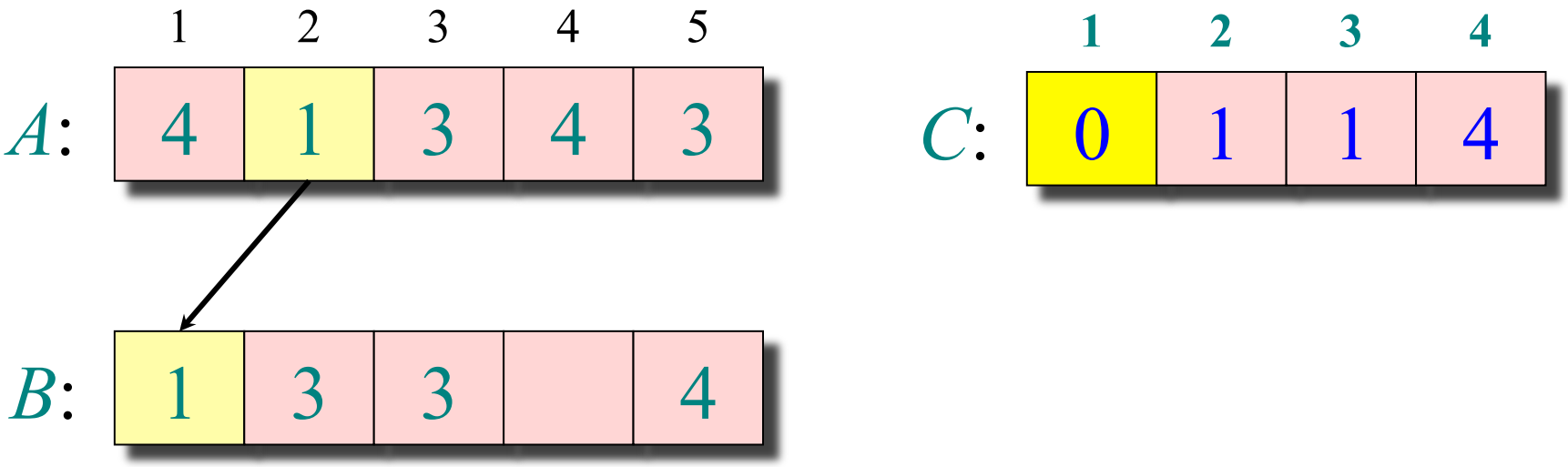
for $j \leftarrow n$ **downto** 1
    do $B[C[A[j]]] \leftarrow A[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A



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

Loop 4: permute elements of A

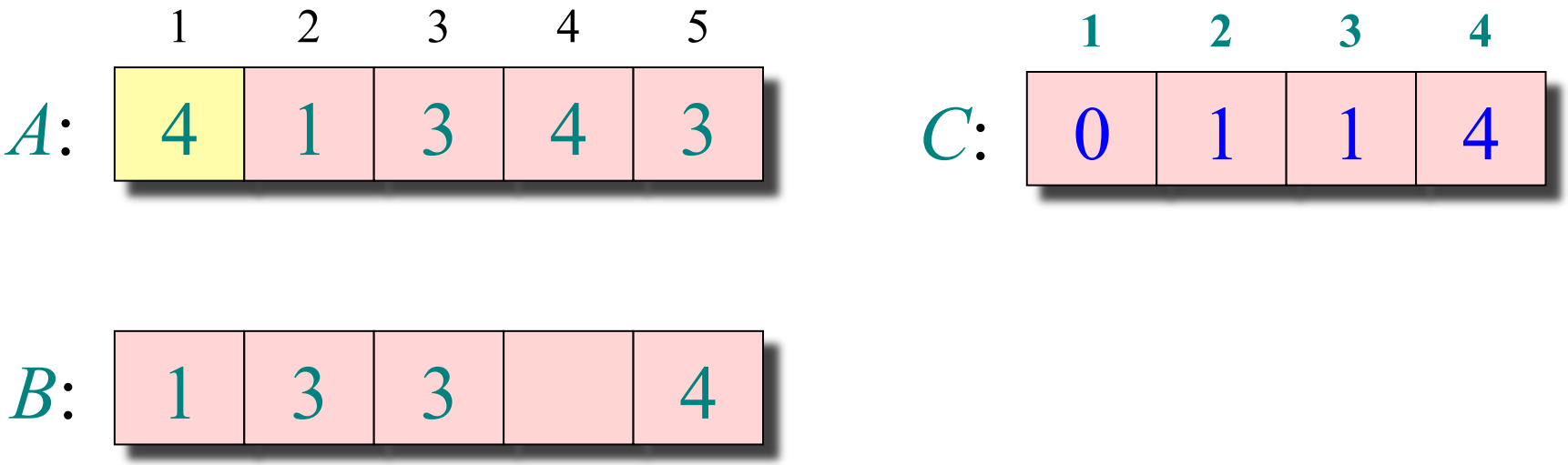


for $j \leftarrow n$ **downto** 1

do $B[C[A[j]]] \leftarrow A[j]$

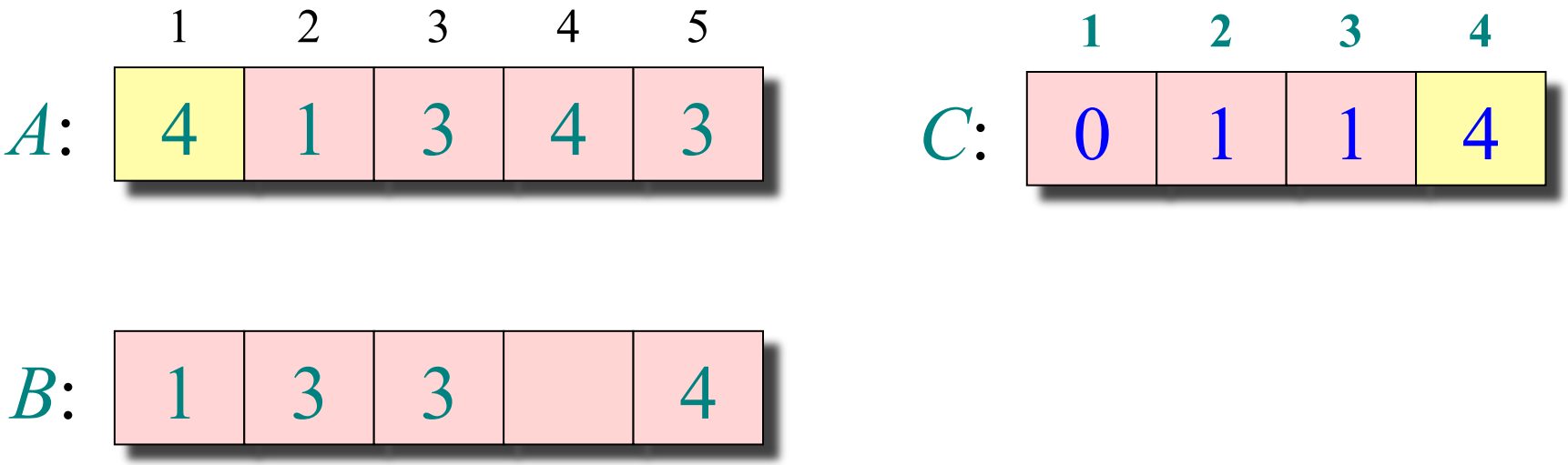
$C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A



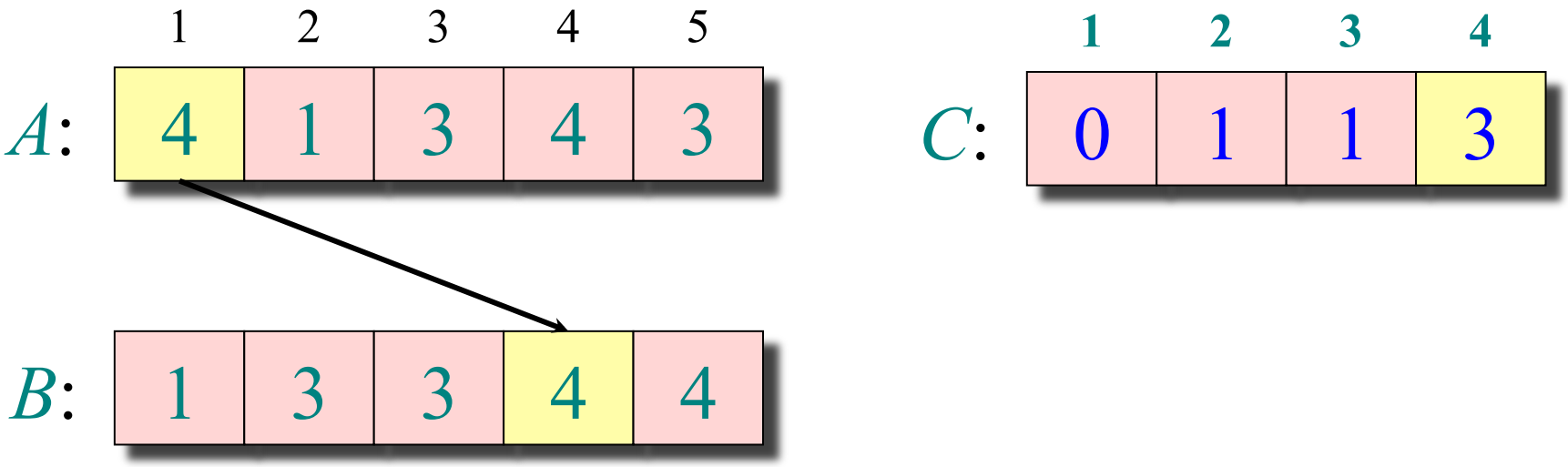
for $j \leftarrow n$ **downto** 1
    do $B[C[A[j]]] \leftarrow A[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A



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

Loop 4: permute elements of A



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

Counting sort

```
for i ← 1 to k
    do C[i] ← 0
for j ← 1 to n
    do C[A[j]] ← C[A[j]] + 1
```

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

$\Theta(k)$

```
for i ← 2 to k
    do C[i] ← C[i] + C[i–1]
```

store in C the cumulative frequencies of different keys in A, i.e. $C[i] = |\{\text{key} \le i\}|$ — $\Theta(k)$

```
for j ← n downto 1
    do B[C[A[j]]] ← A[j]
       C[A[j]] ← C[A[j]] – 1
```

using cumulative frequencies build sorted permutation — $\Theta(n)$

$\Theta(n + k)$

Running time

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

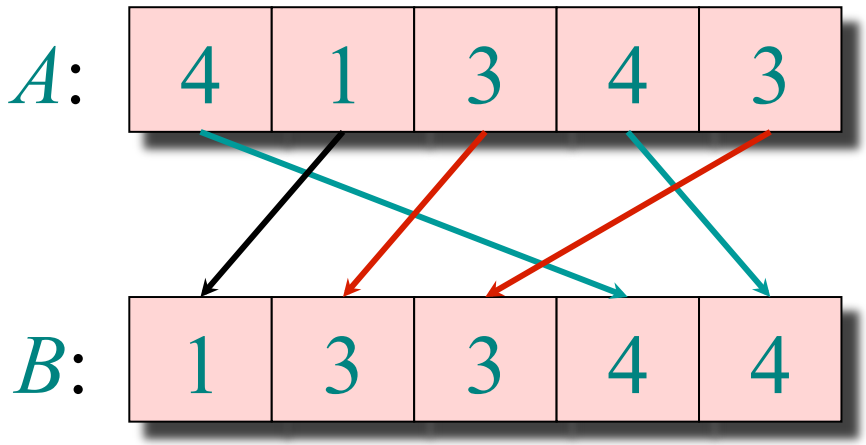
- But, sorting takes $\Omega(n \lg n)$ time!
- Where's the fallacy?

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.

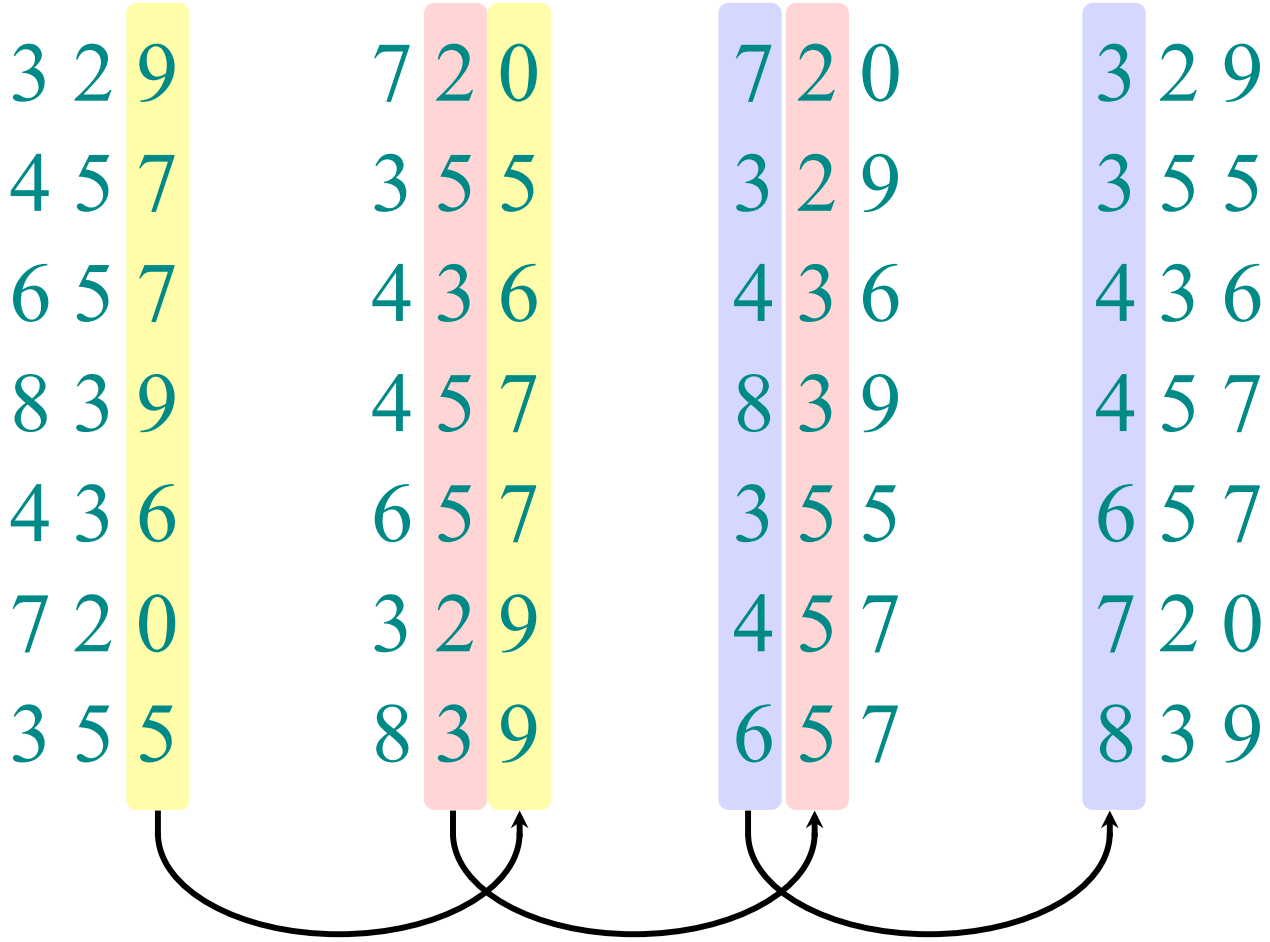


This does not seem useful for this example, but imagine a situation where each element stored in A comes with some "personalized information" (wait 2 slides…).

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.

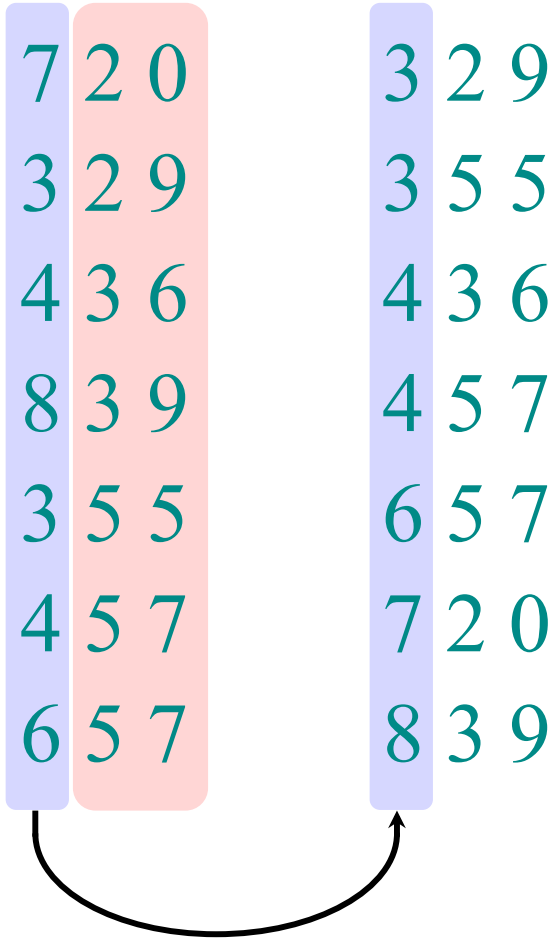
Operation of radix sort



Correctness of radix sort

Induction on digit position

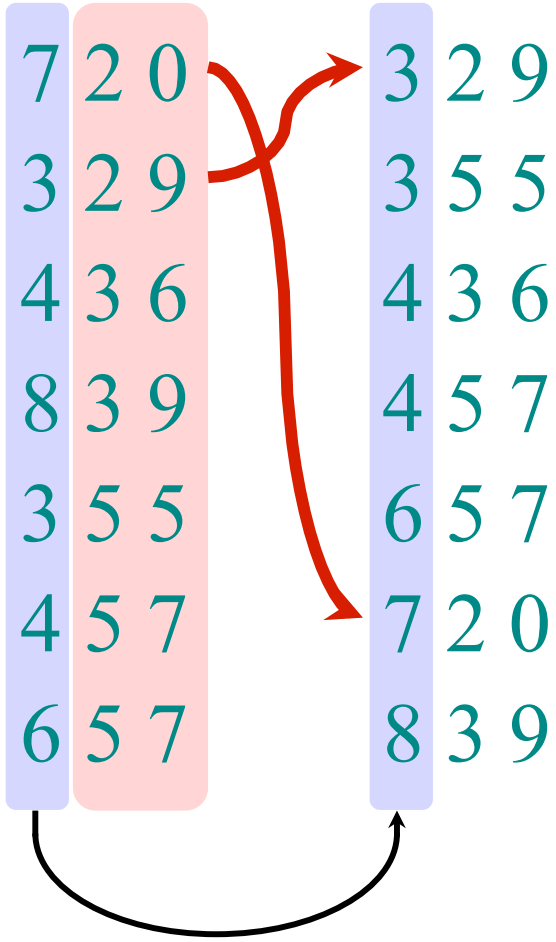
- Assume that the numbers are sorted by their low-order $t - 1$ digits.
- Sort on digit t



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.

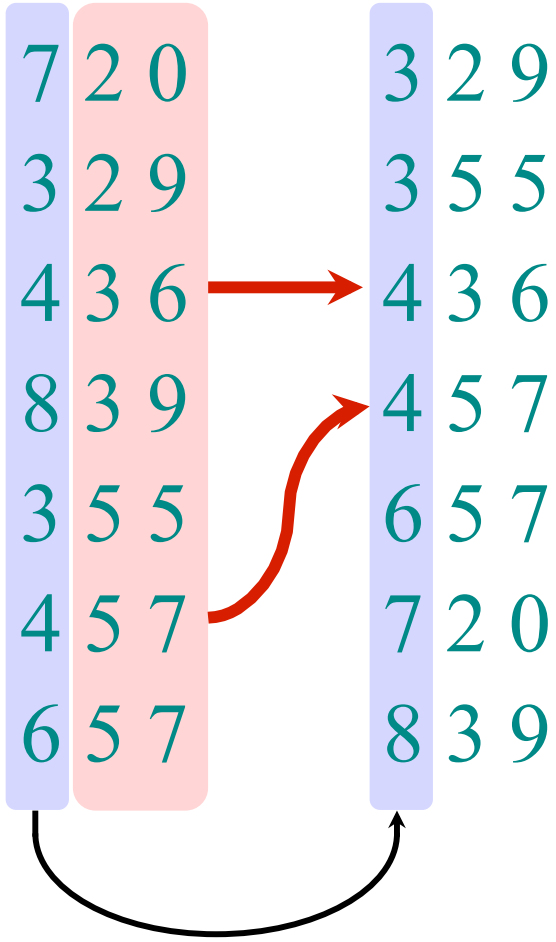


Correctness of radix sort

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.

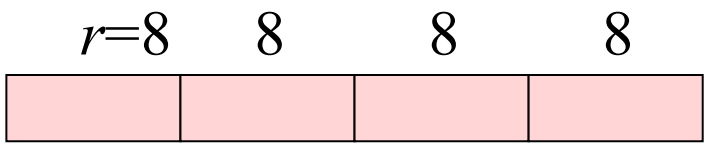
(just used stability property!)



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: b=32-bit word

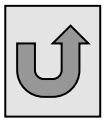


- If each b-bit word is broken into r-bit pieces, each pass of counting sort takes $\Theta(n + 2^r)$ time.
- So overall $\Theta(b/r\ (n + 2^r))$ time.
- Setting $r=\log n$ gives $\Theta(n)$ time per pass, or $\Theta(n\ b/\log n)$ total

Appendix: Punched-card technology

- Herman Hollerith (1860-1929)
- Punched cards
- Hollerith's tabulating system
- Operation of the sorter
- Origin of radix sort
- "Modern" IBM card
- Web resources on punched-card technology

Return to last slide viewed.

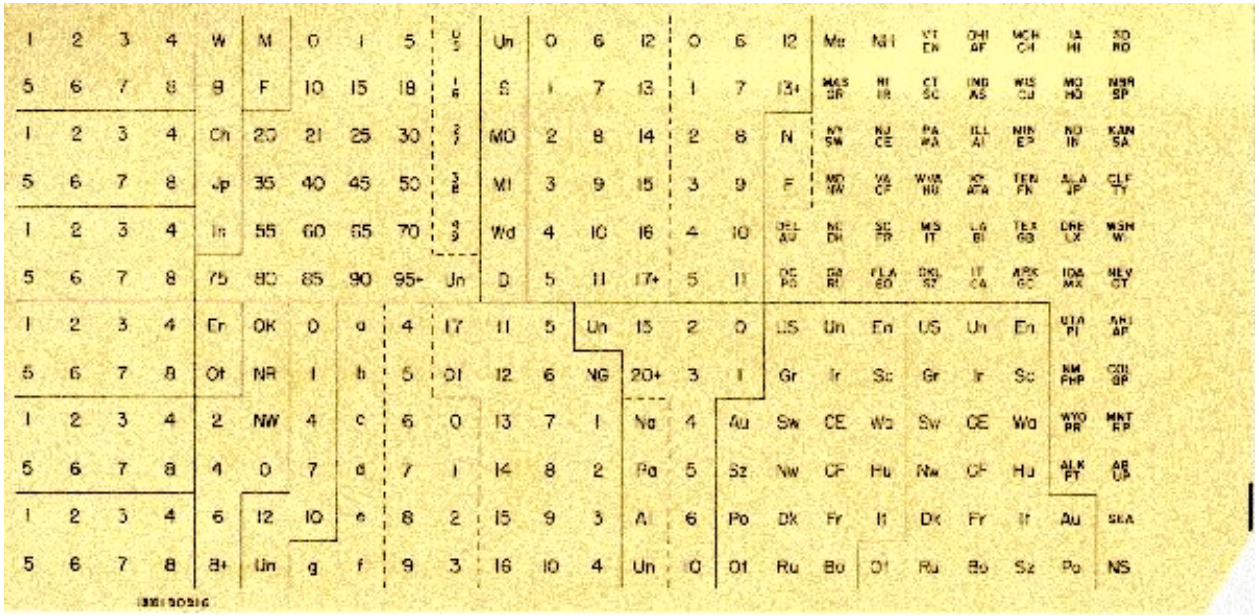


Herman Hollerith (1860-1929)

- 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

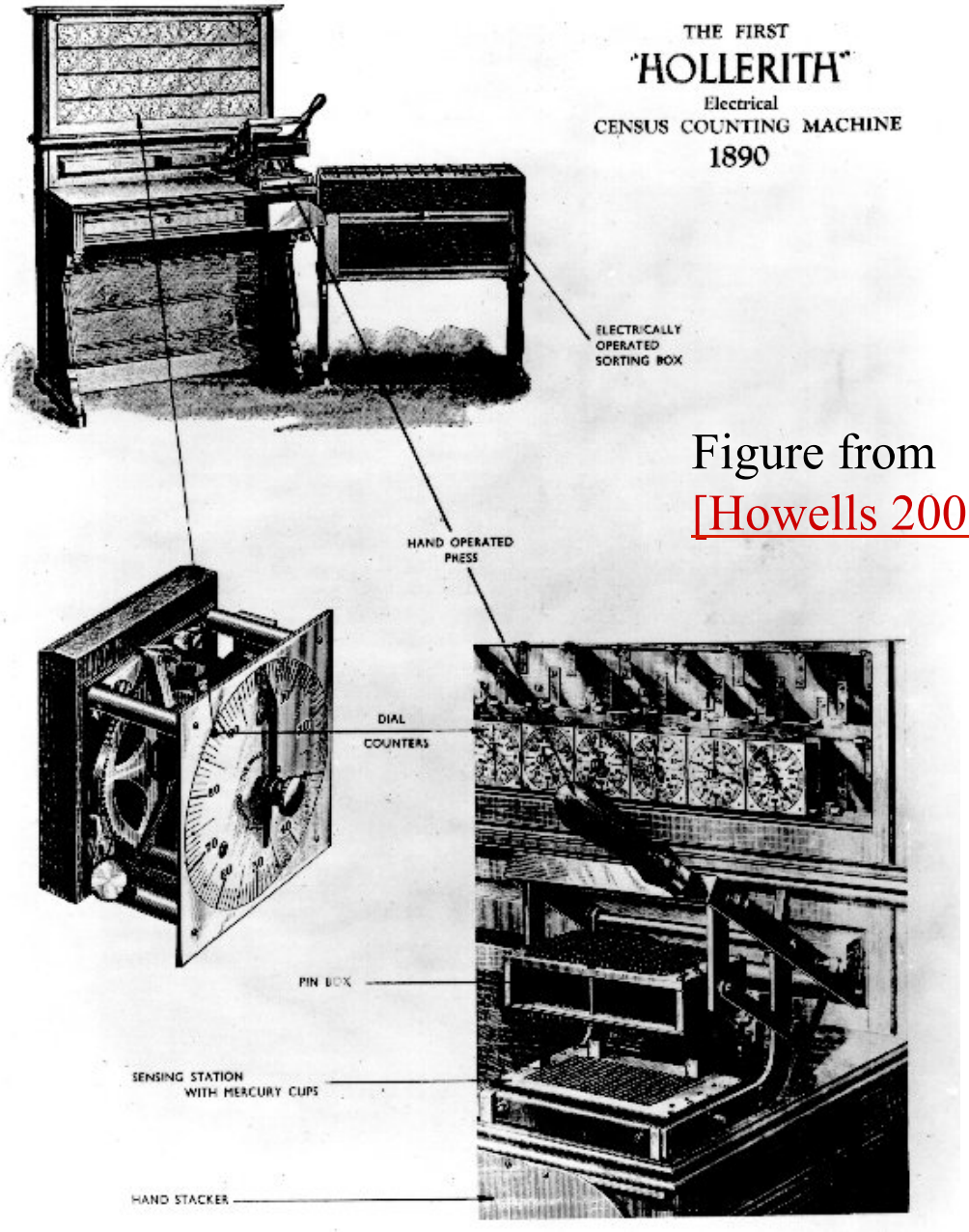
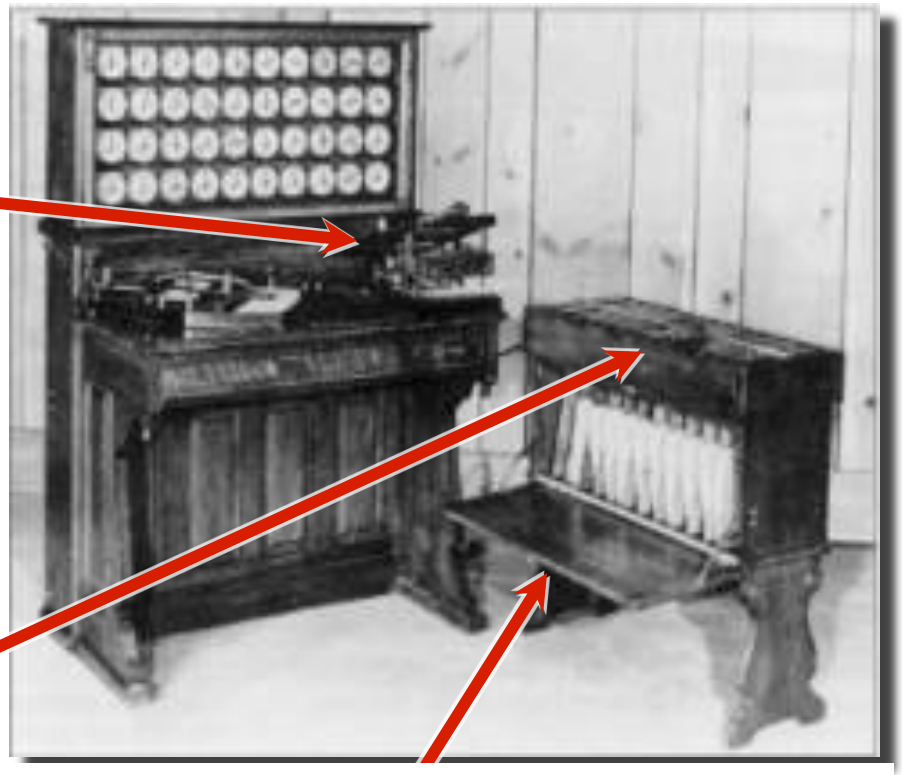


Figure from [Howells 2000].

Operation of the 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

Origin of radix sort

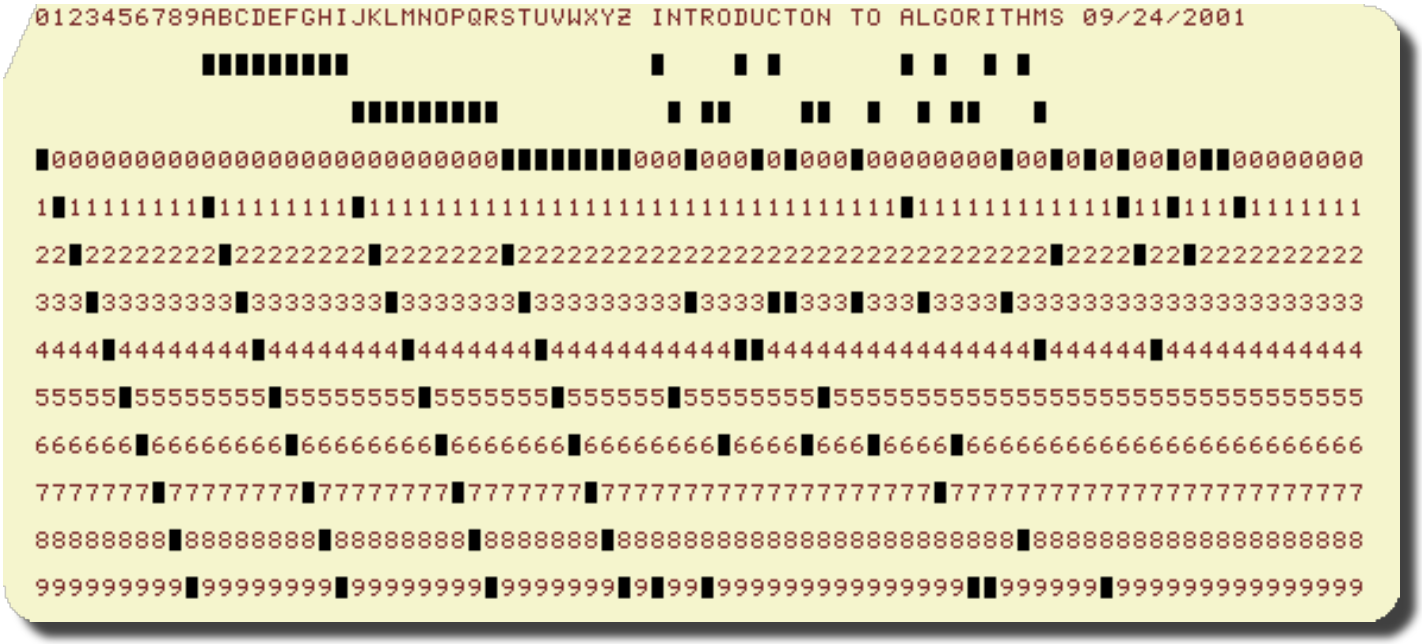
Hollerith's original 1889 patent alludes to a most-significant-digit-first radix sort:

> *"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.

“Modern” IBM card

- One character per column.



Produced by the WWW Virtual Punch-Card Server.

So, that’s why text windows have 80 columns!

Web resources on punched-card technology

- Doug Jones's punched card index
- Biography of Herman Hollerith
- The 1890 U.S. Census
- Early history of IBM
- Pictures of Hollerith's inventions
- Hollerith's patent application (borrowed from Gordon Bell's CyberMuseum)
- Impact of punched cards on U.S. history