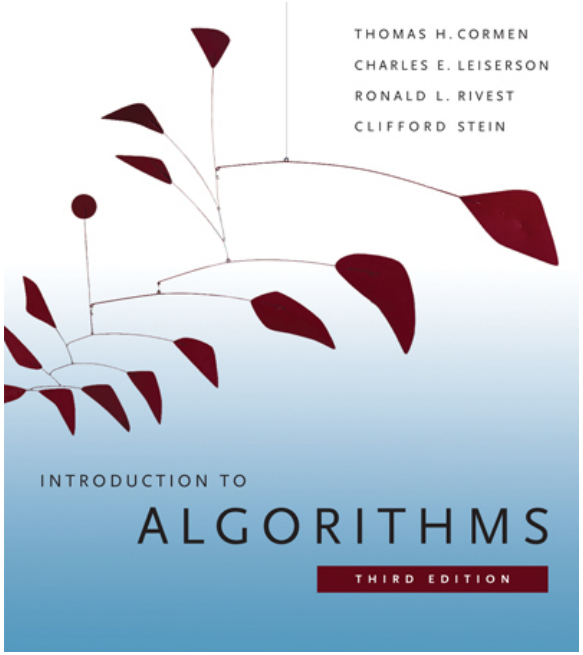
6.006- *Introduction to Algorithms*



Lecture 10

Prof. Piotr Indyk

Quiz Rules

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page
- When the quiz begins, write your name on every page of this quiz booklet.
- You have 120 minutes to earn 120 points. Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress.
- This quiz booklet contains X pages, including this one. Two extra sheets of scratch paper are attached. Please detach them before turning in your quiz at the end of the exam period.
- This quiz is closed book. You may use one 8 1 ″ × 11″ or A4 crib sheets (both sides). No calculators or programmable devices are permitted. No cell phones or other communications devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Pages may be separated for grading.
- Do not waste time and paper re-deriving facts that we have studied. It is sufficient to cite known results.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.

*At participating locations only. Do not use while sleeping. Read label before using. Details inside.Harmful if swallowed.

Menu: sorting ctd.

- 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.*, insertion sort, merge sort, heapsort.

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

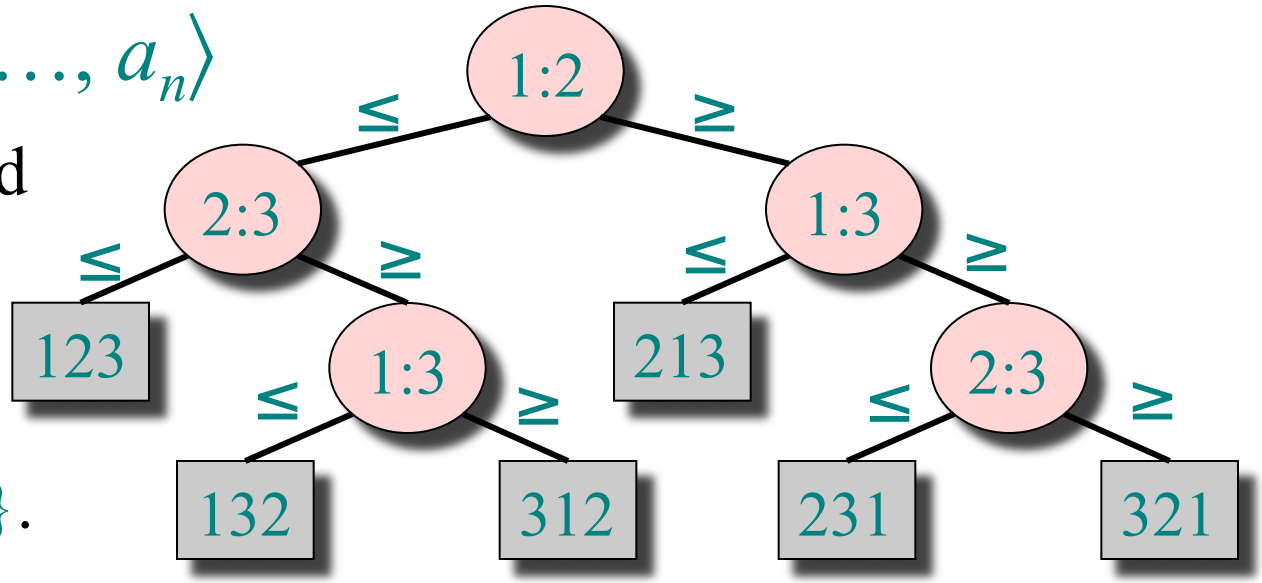
Is $O(n \lg n)$ the best we can do via comparisons?

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 $i, j \in \{1, 2, \ldots, n\}$.

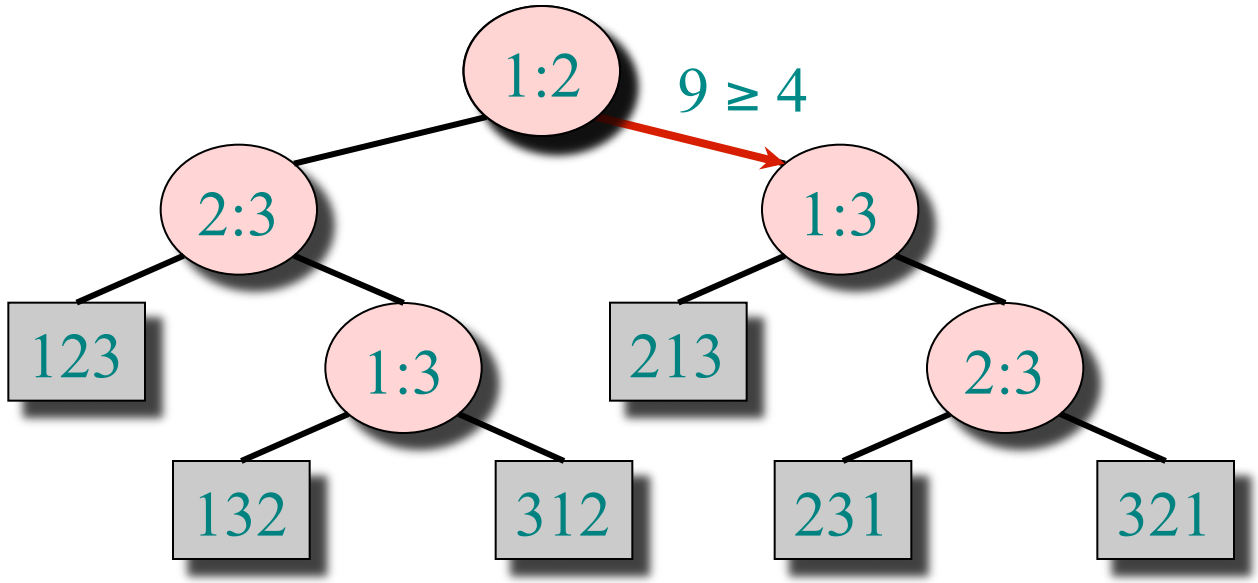


- Branching direction depends on outcome of comparisons.

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

Decision-tree example

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

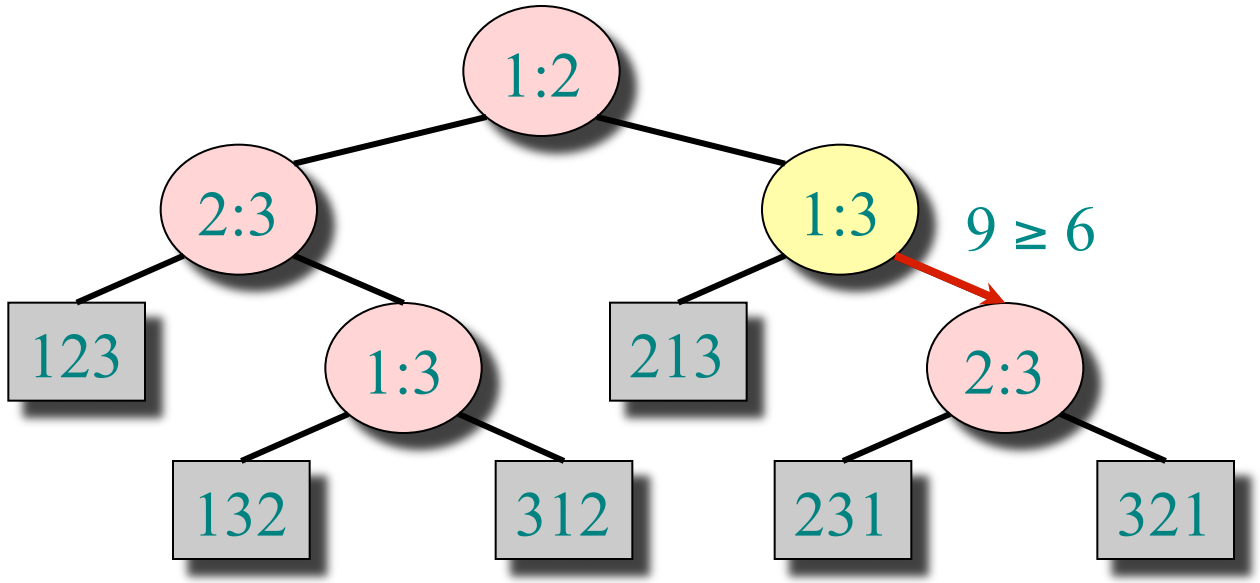


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$.

Decision-tree example

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

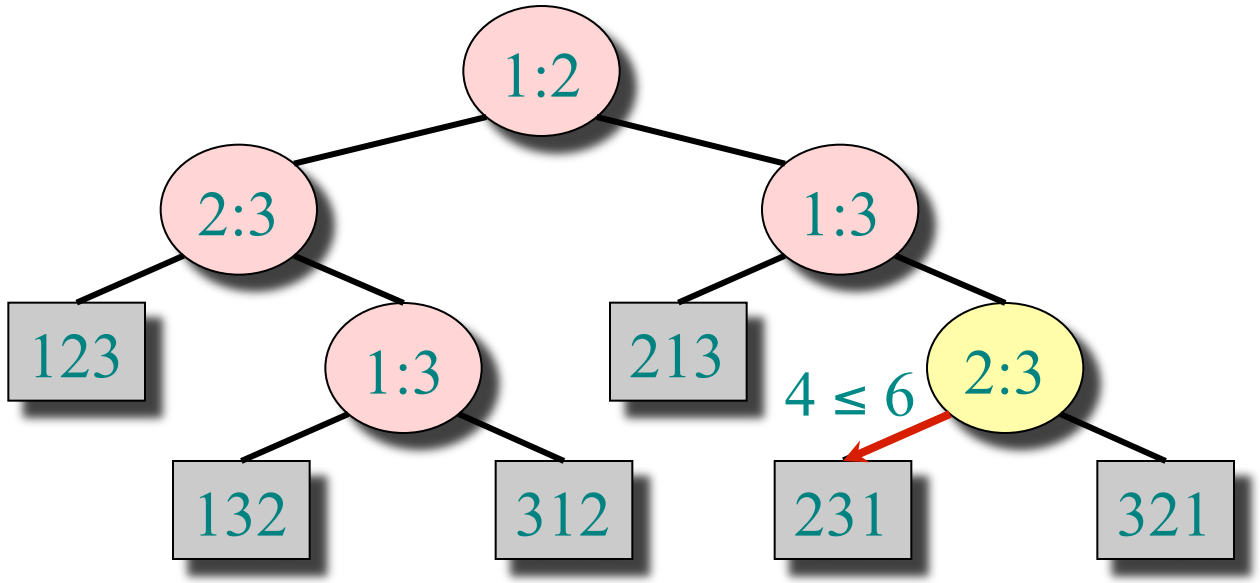


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

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

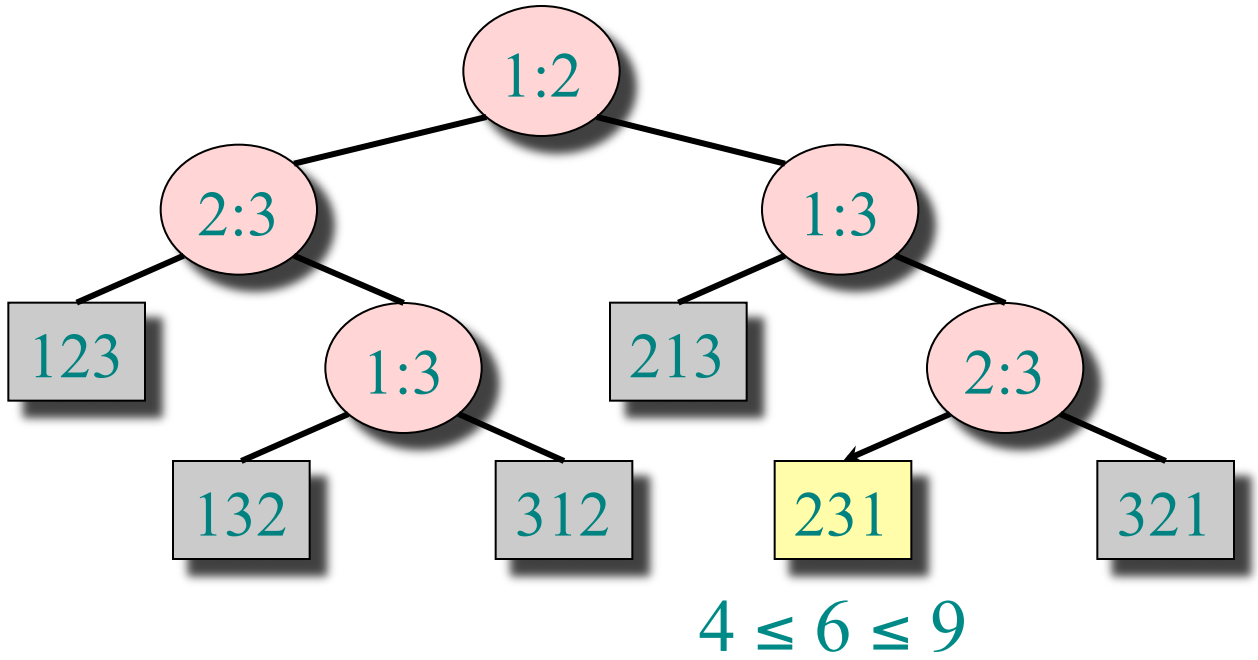


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- 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 $a_{\pi(1)} \le a_{\pi(2)} \le \cdots \le a_{\pi(n)}$ 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

- 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

(implicitly, we also determine the permutation π transforming A into B)

- ***Auxiliary storage***: $C[1 \,..\, k]$.

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. $C[i] = |\{\text{key} = i\}|$

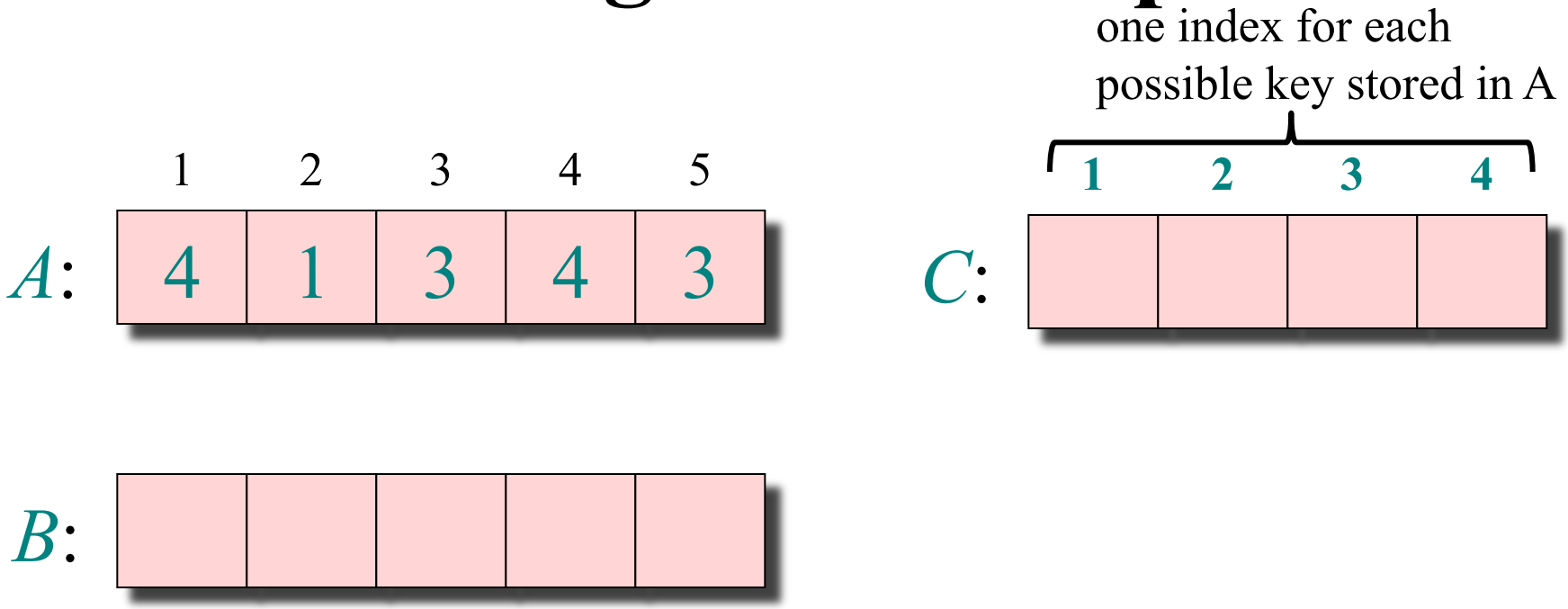
```
for i ← 2 to k
    do C[i] ← C[i] + C[i–1]
```

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

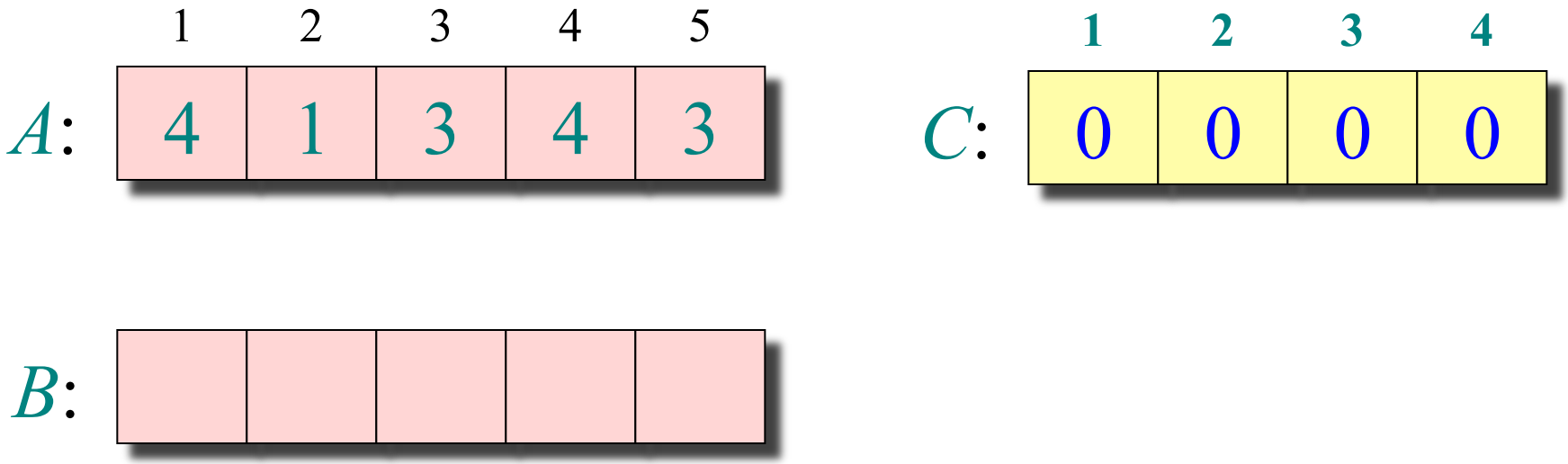
```
for j ← n downto 1
    do B[C[A[j]]] ← A[j]
       C[A[j]] ← C[A[j]] – 1
```

using cumulative frequencies build sorted permutation

Counting-sort example



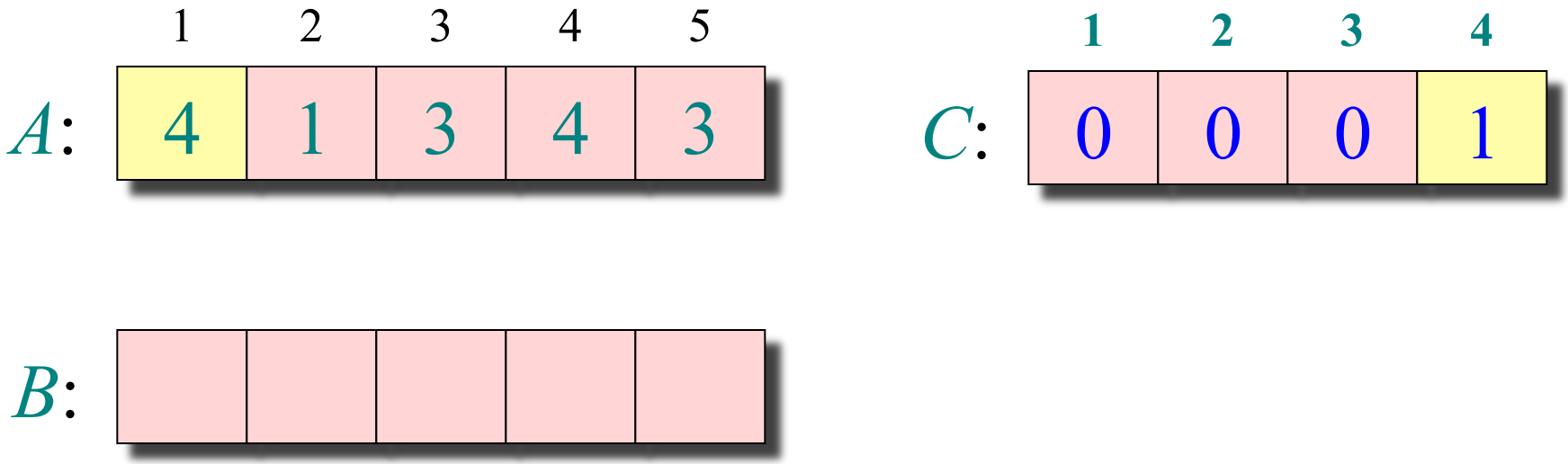
Loop 1: initialization



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

do $C[i] \leftarrow 0$

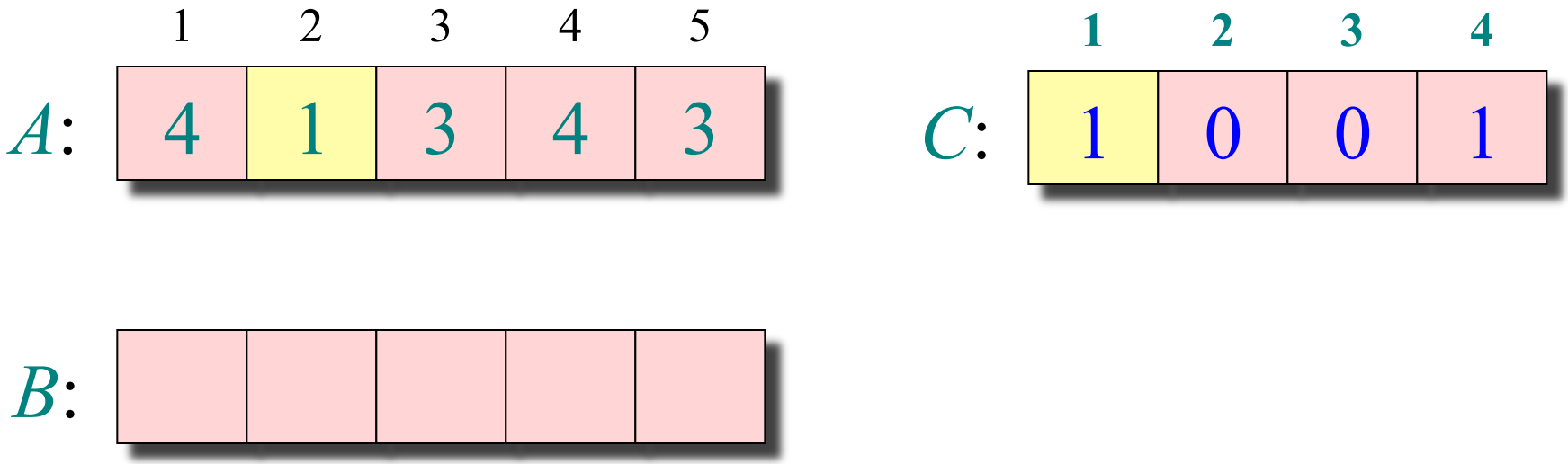
Loop 2: count frequencies



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

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

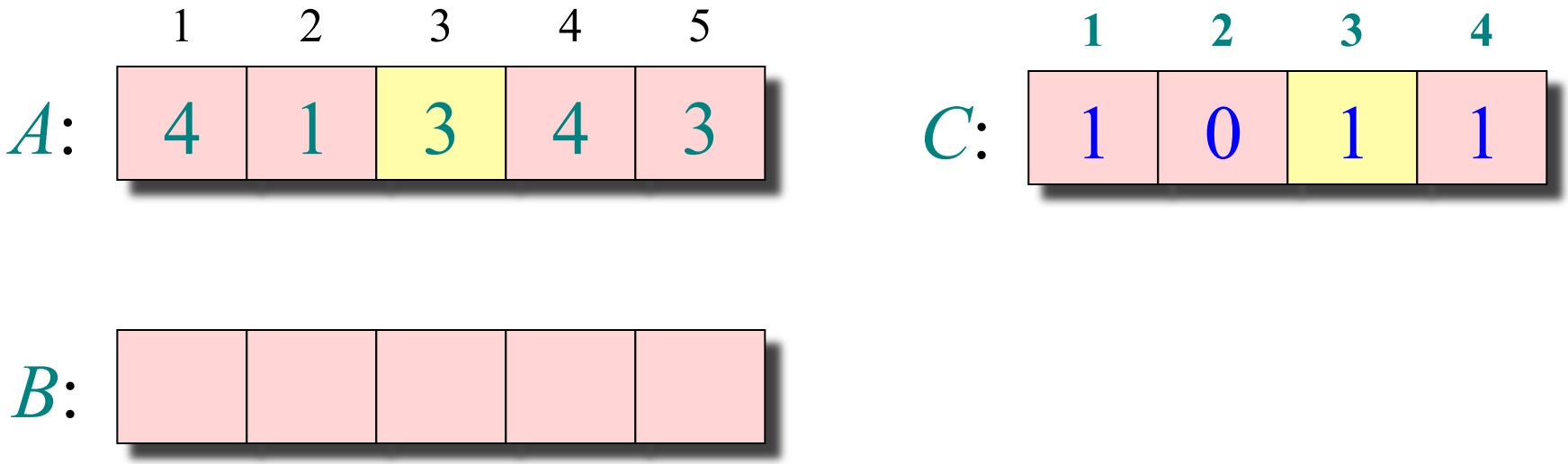
Loop 2: count frequencies



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

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

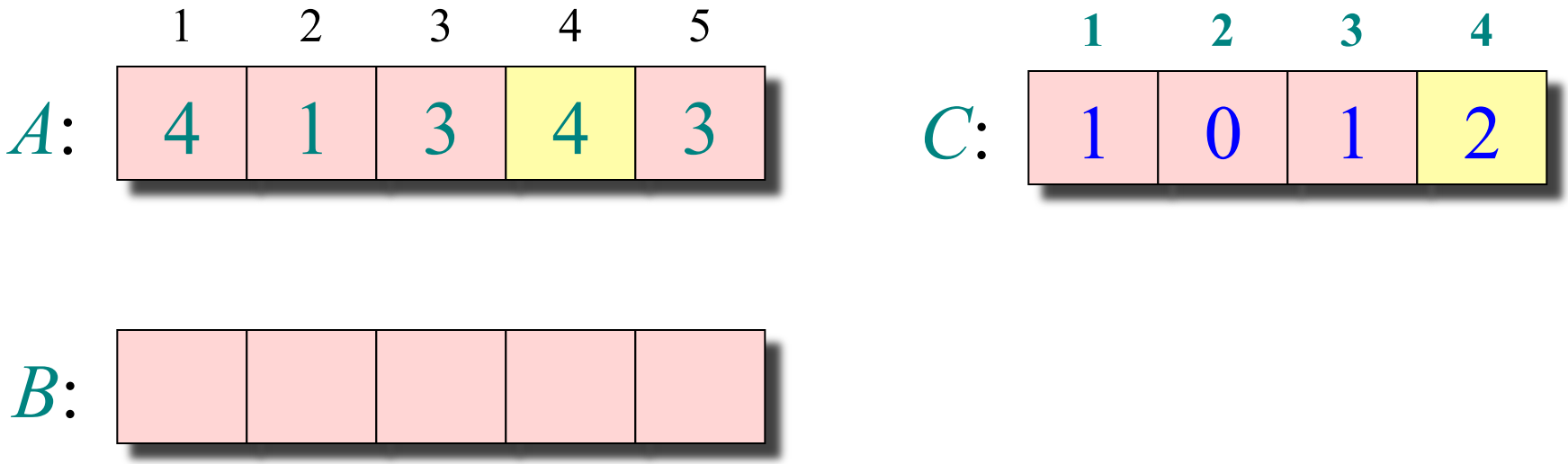
Loop 2: count frequencies



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

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

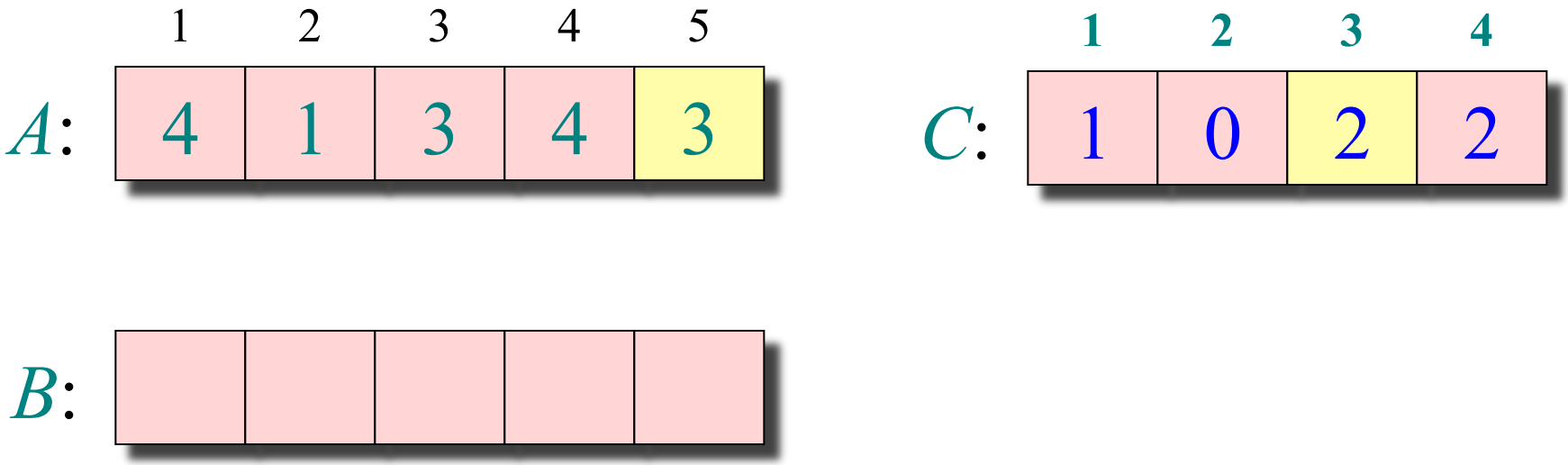
Loop 2: count frequencies



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

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

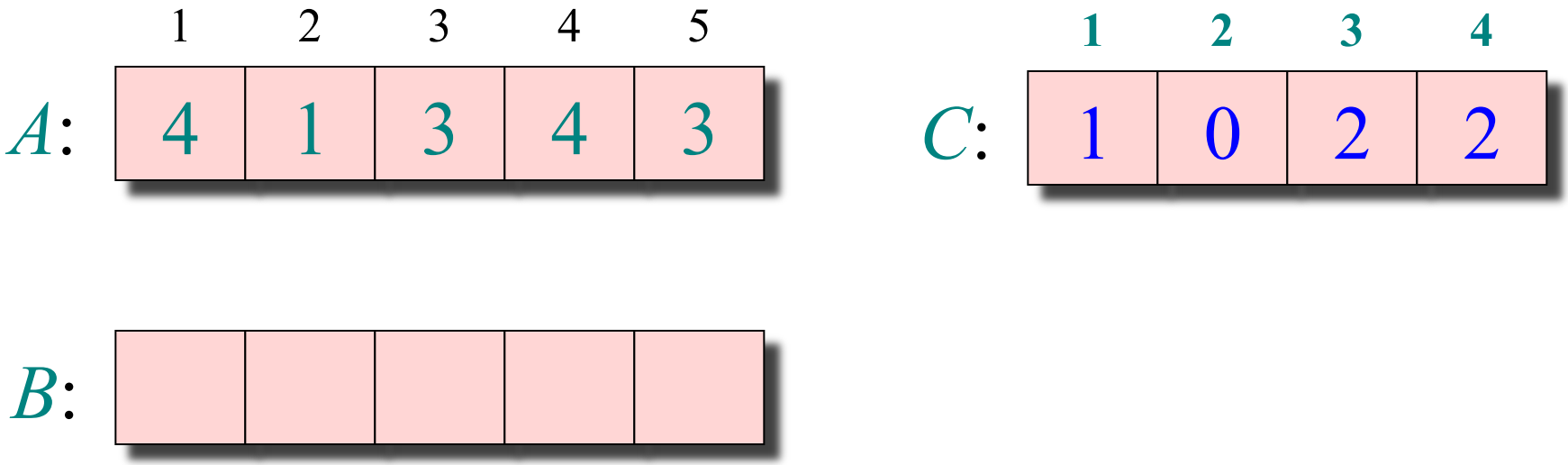
Loop 2: count frequencies



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

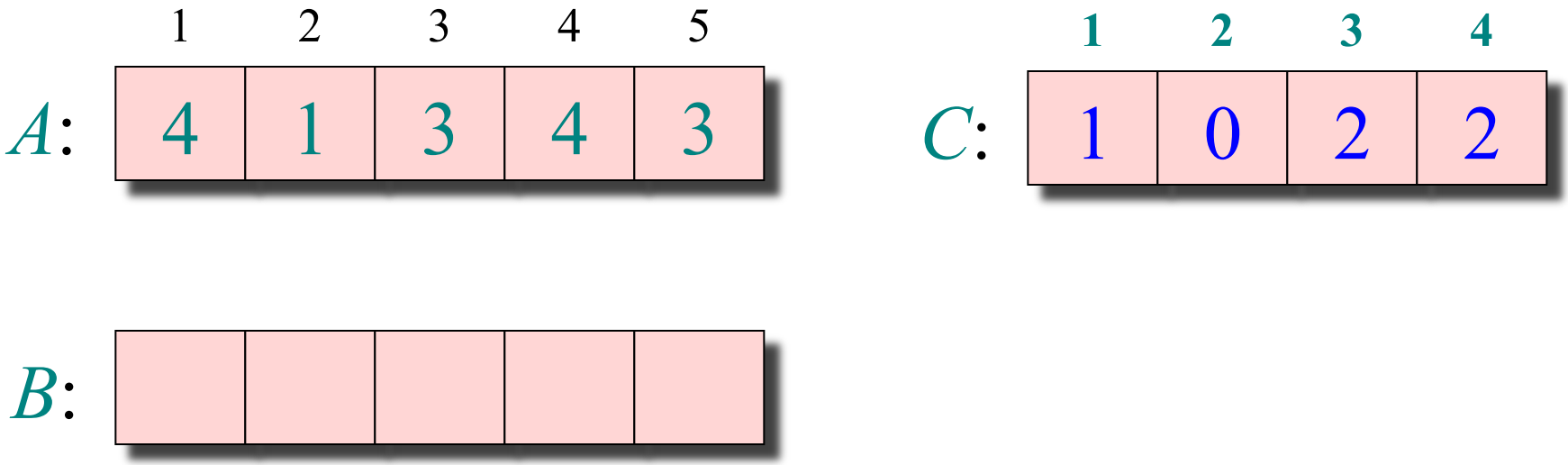
do $C[A[j]] \leftarrow C[A[j]] + 1$ ▷ $C[i] = |\{\text{key} = i\}|$

Loop 2: count frequencies



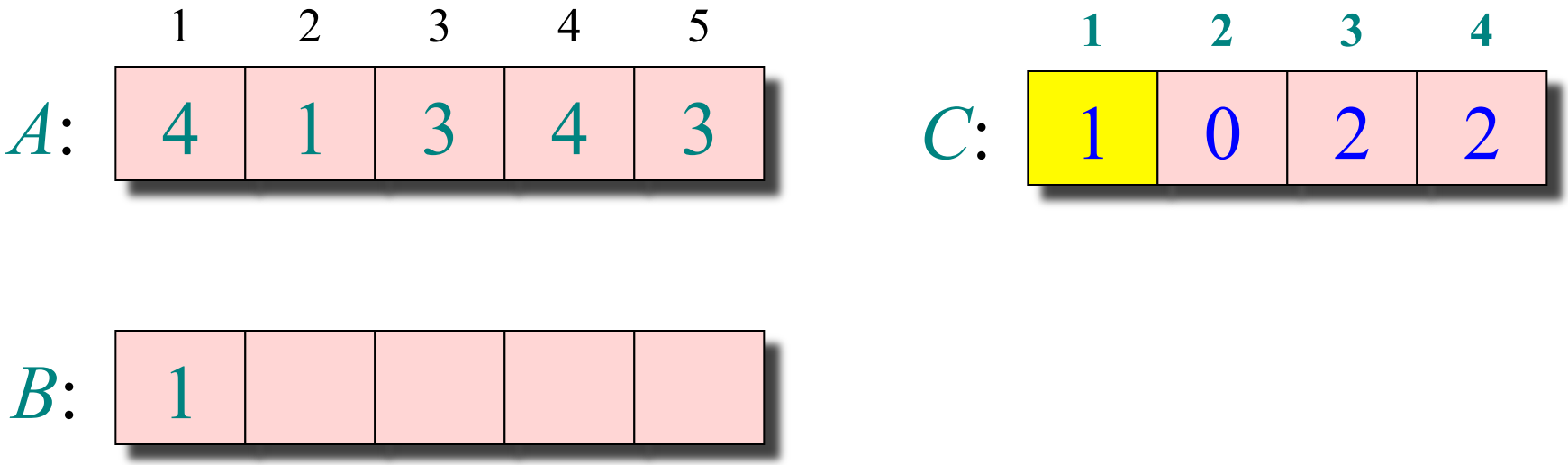
for $j \leftarrow 1$ **to** n

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

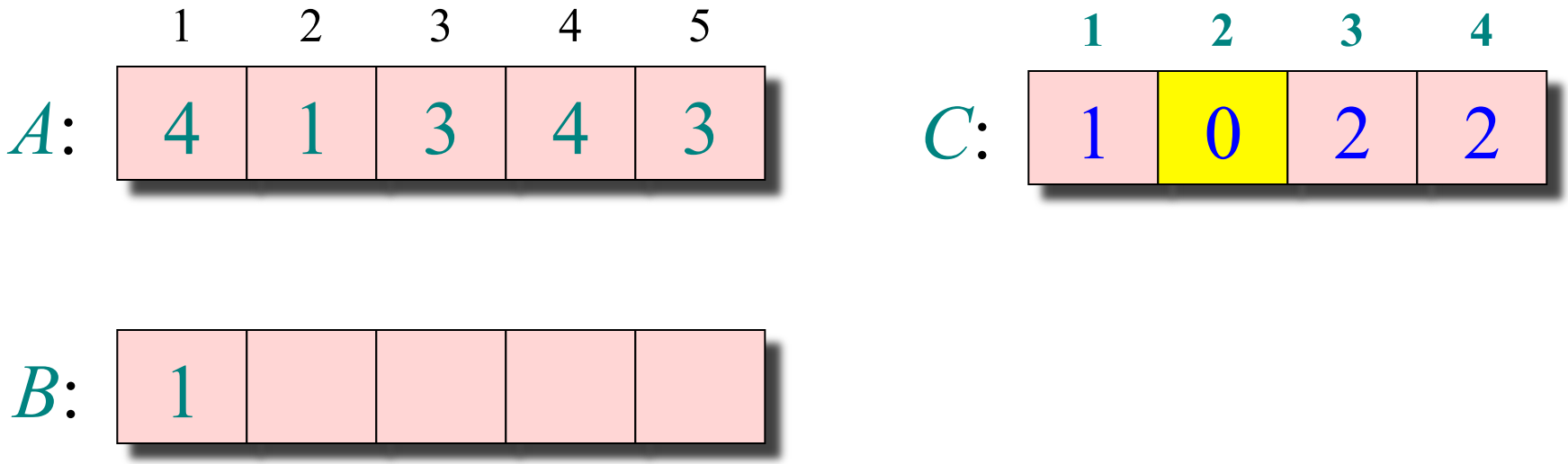


Walk through frequency array an place the appropriate number of each key in output array…

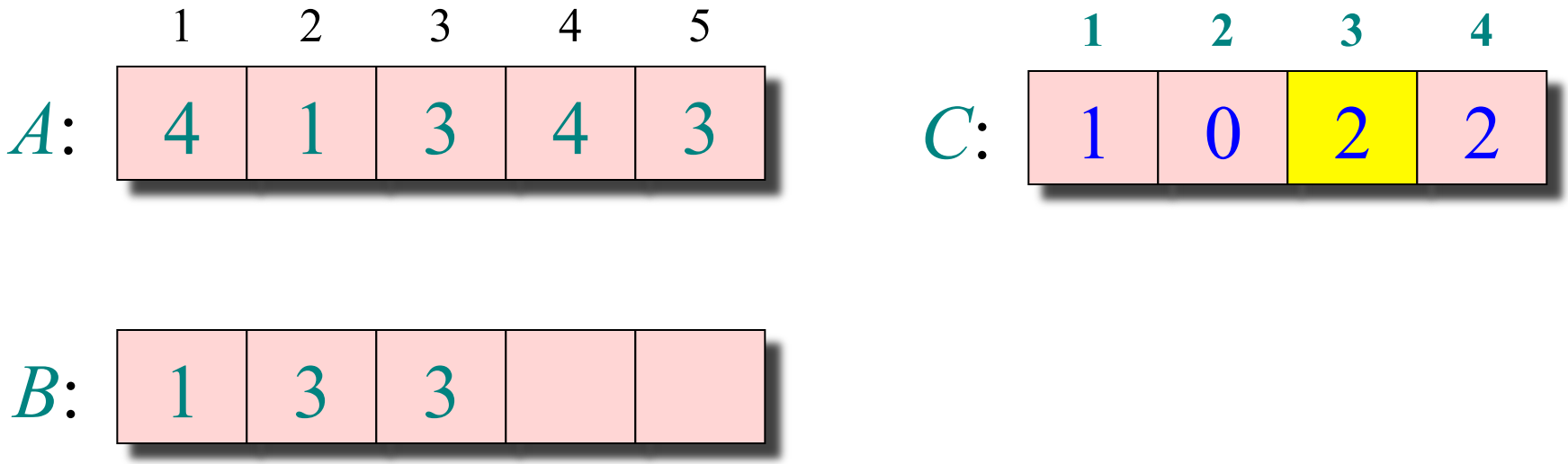
A parenthesis: a quick finish



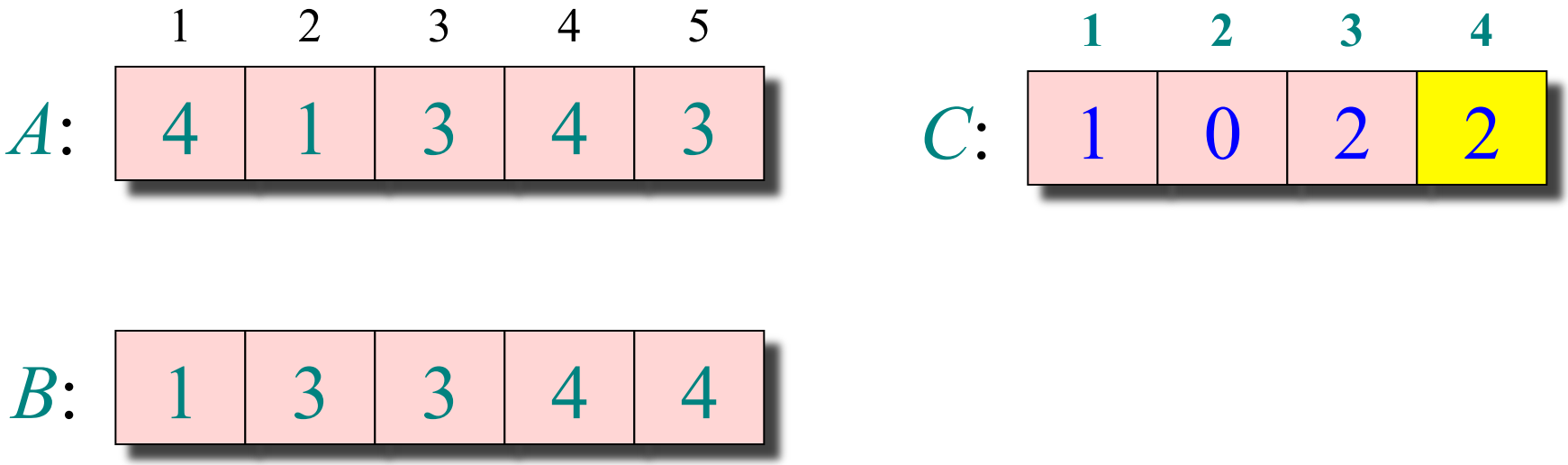
A parenthesis: a quick finish



A parenthesis: a quick finish



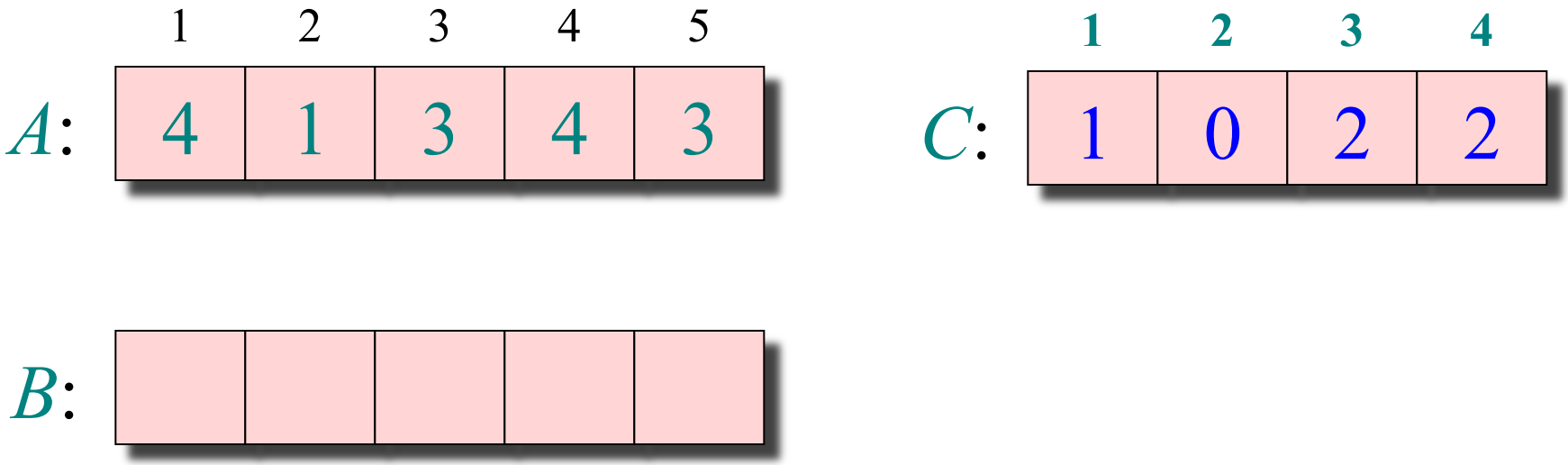
A parenthesis: a quick finish



B is sorted!

but no permutation π

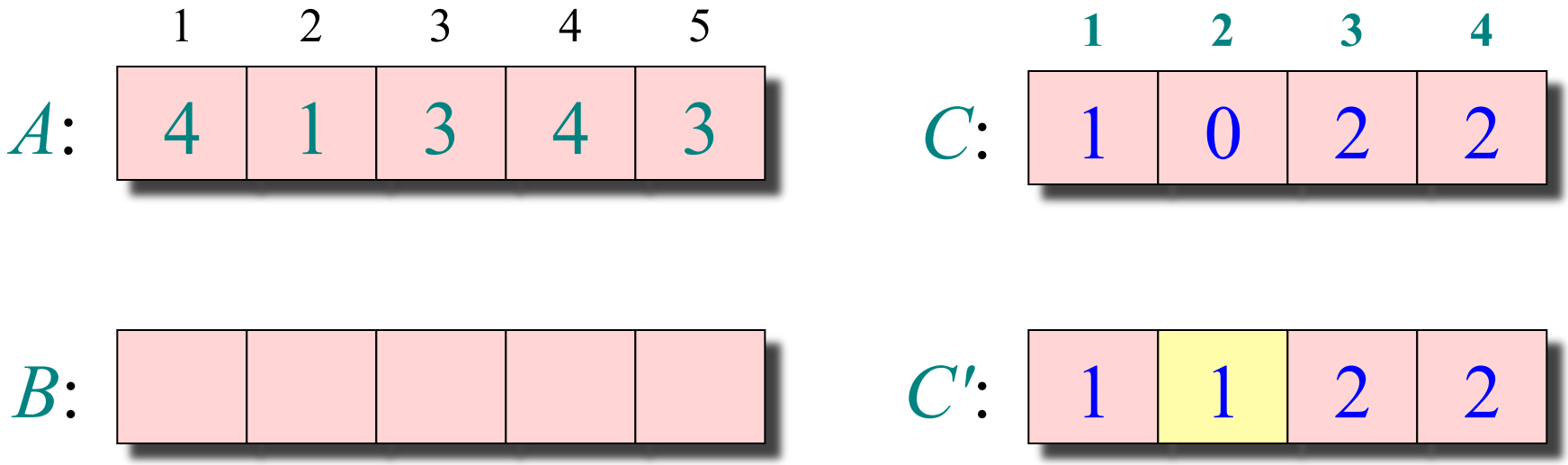
Loop 2: count frequencies



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

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

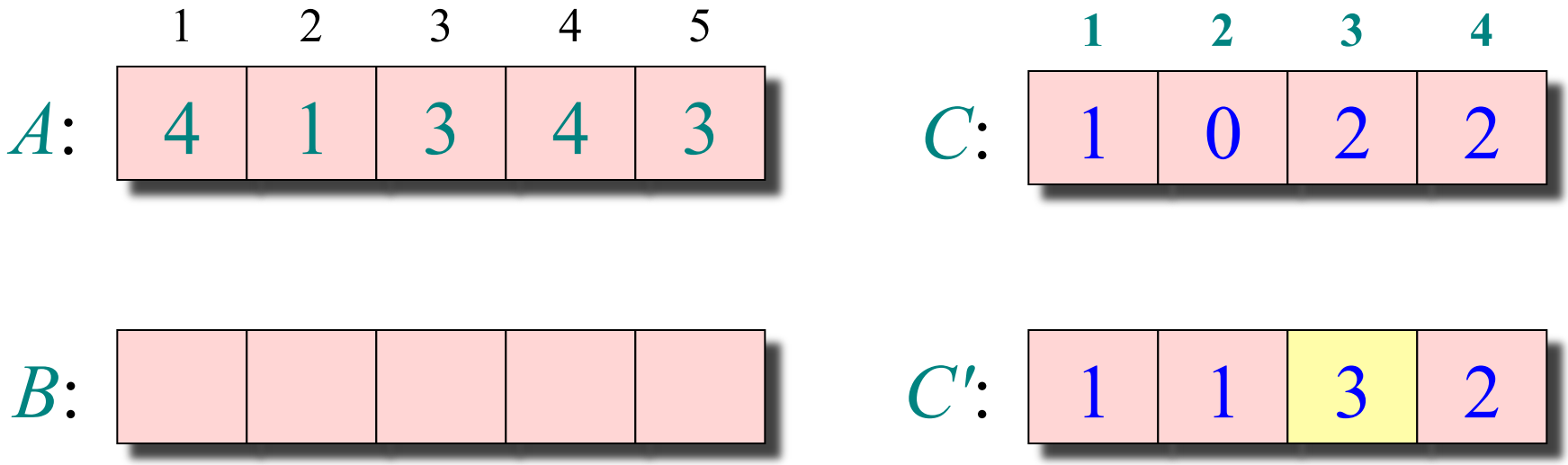
Loop 3: cumulative frequencies



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

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

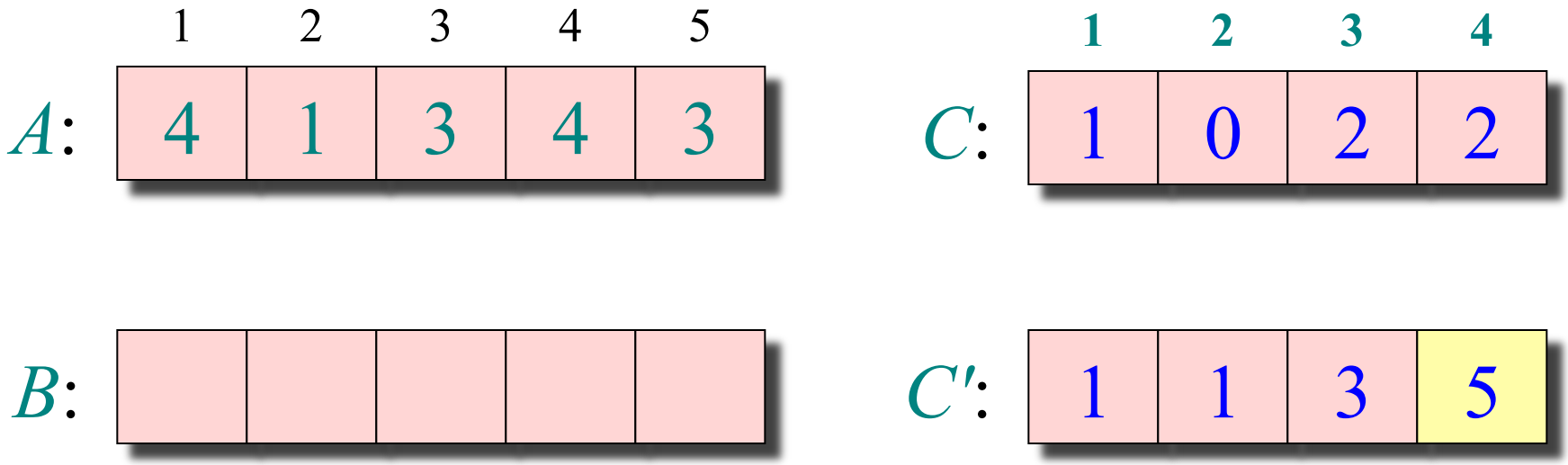
Loop 3: cumulative frequencies



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

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

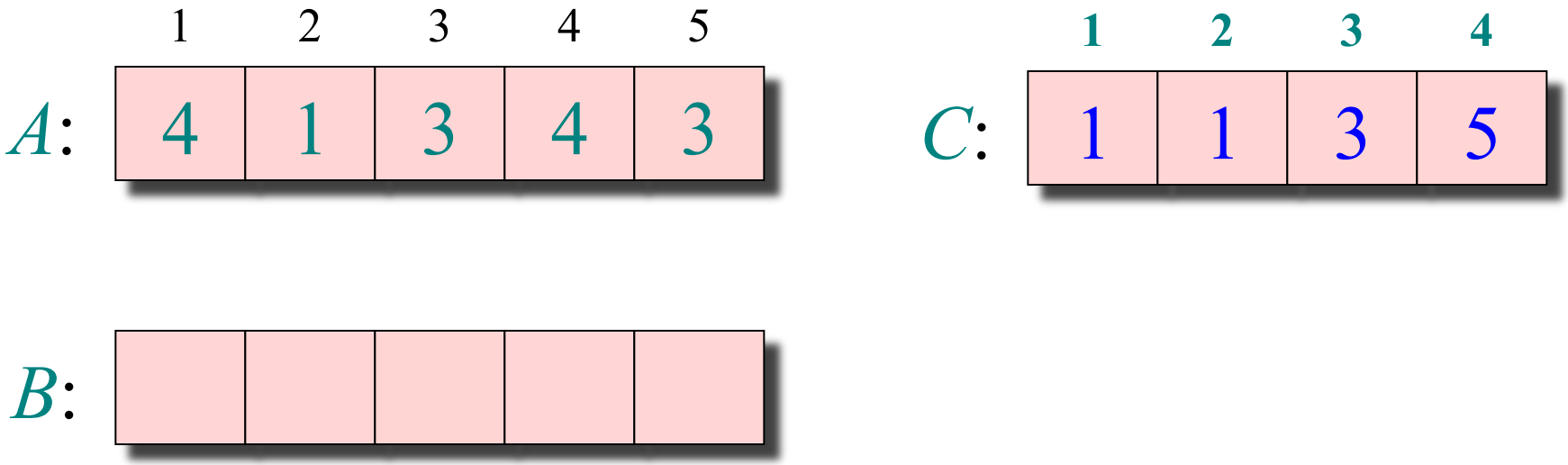
Loop 3: cumulative frequencies



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

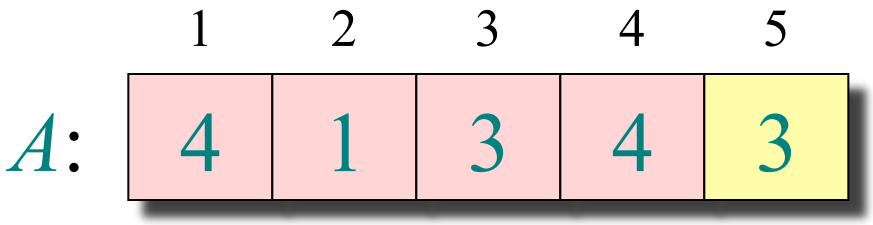


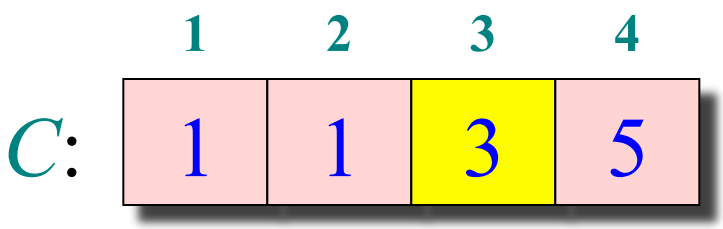
for $j \leftarrow n$ **downto** 1

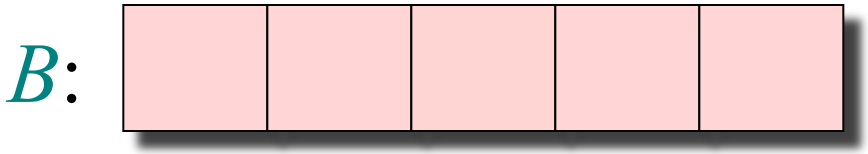
do $B[C[A[j]]] \leftarrow A[j]$

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

Loop 4: permute elements of A



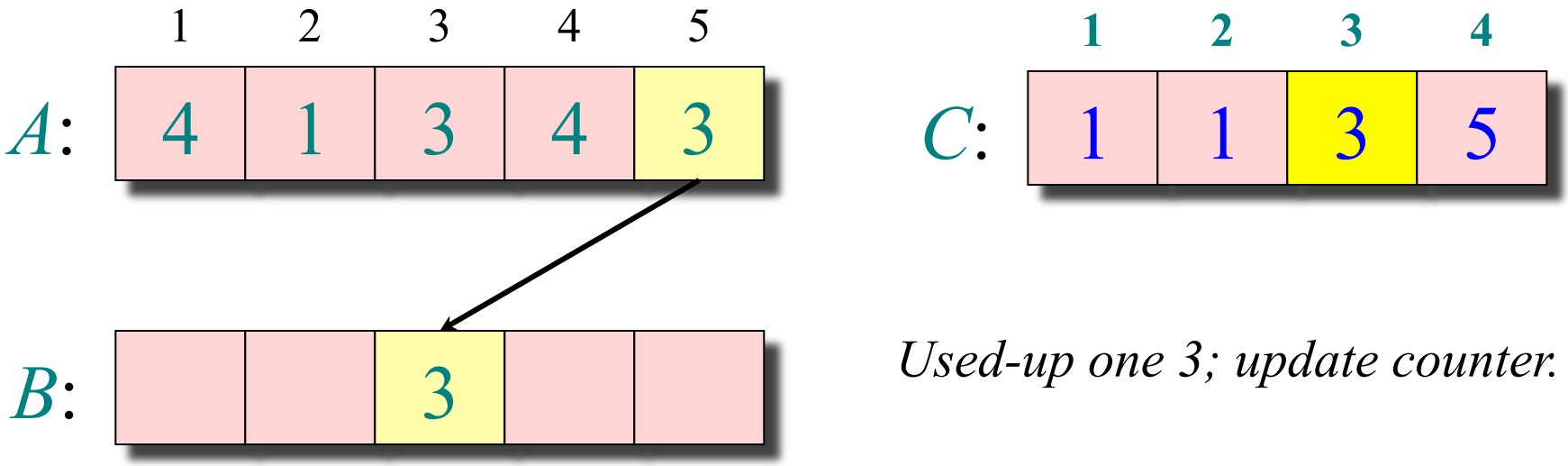




There are exactly 3 elements ≤A[5]; so where should I place A[5]?

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

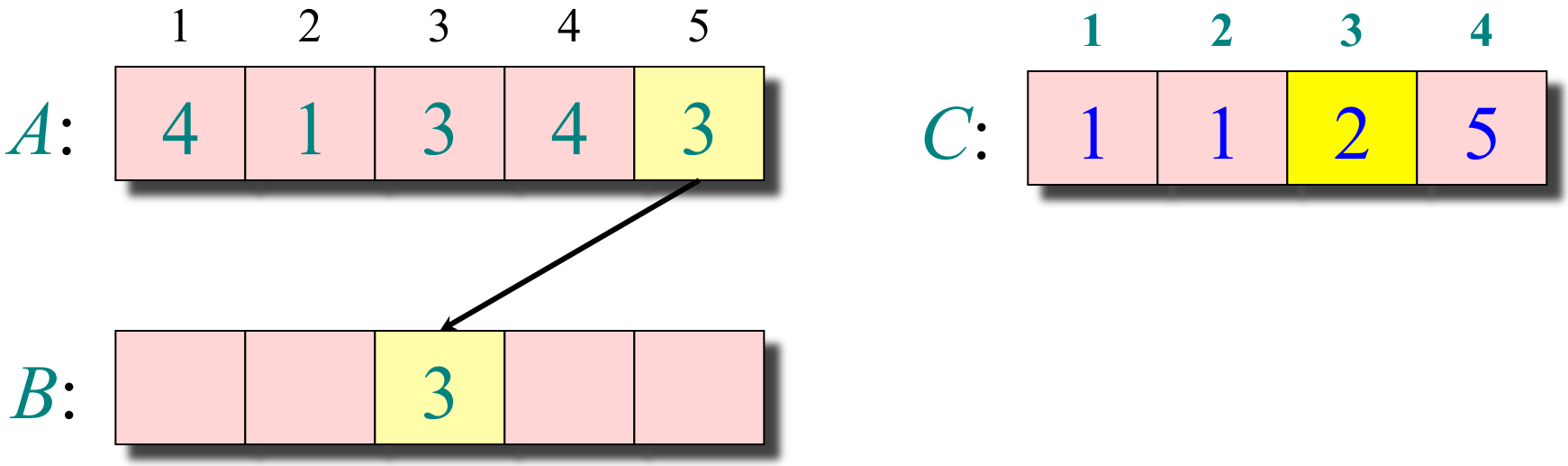
Loop 4: permute elements of A



Used-up one 3; update counter.

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

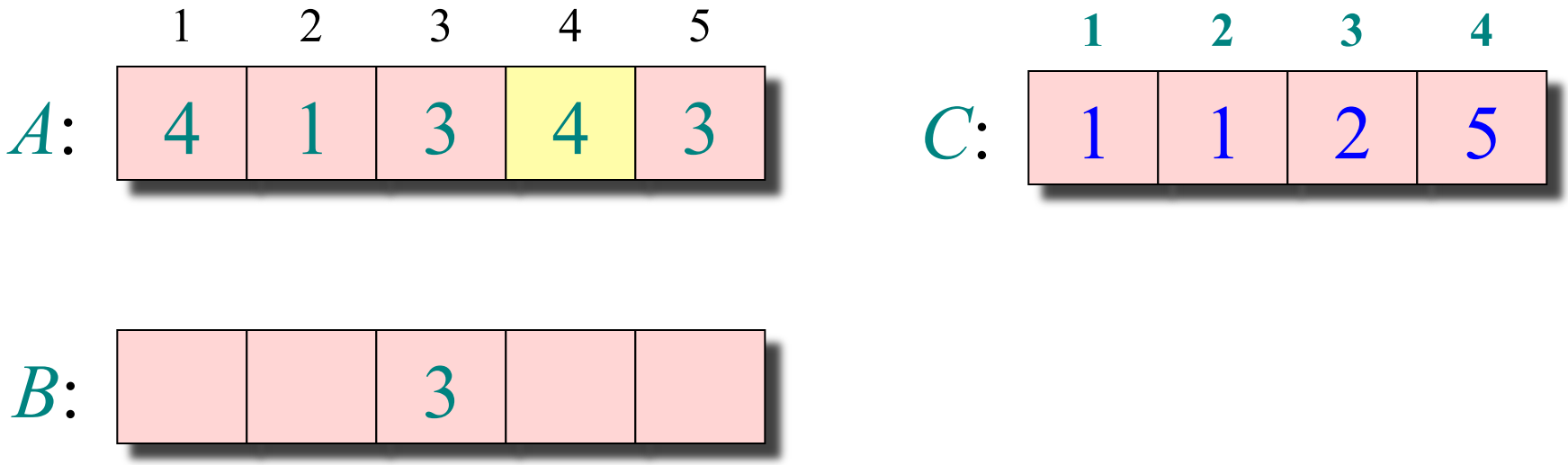


for $j \leftarrow n$ **downto** 1

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

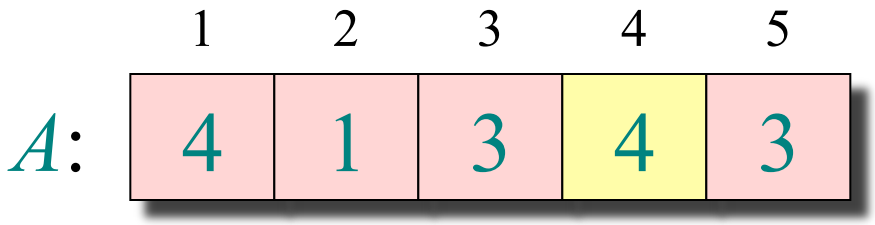
$C[A[j]] \leftarrow C[A[j]] - 1$

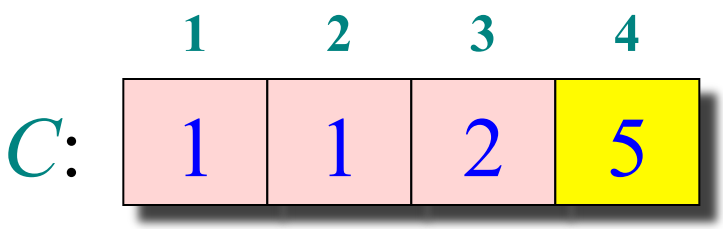
Loop 4: permute elements of A

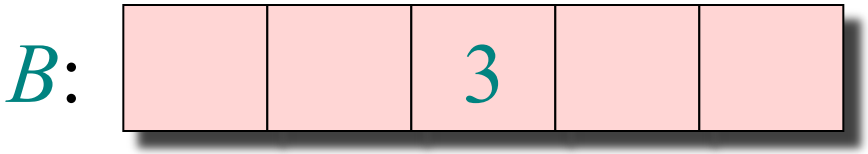


for $j \leftarrow n$ **downto** 1
  do $B[C[A[j]]] \leftarrow A[j]$
    $C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A



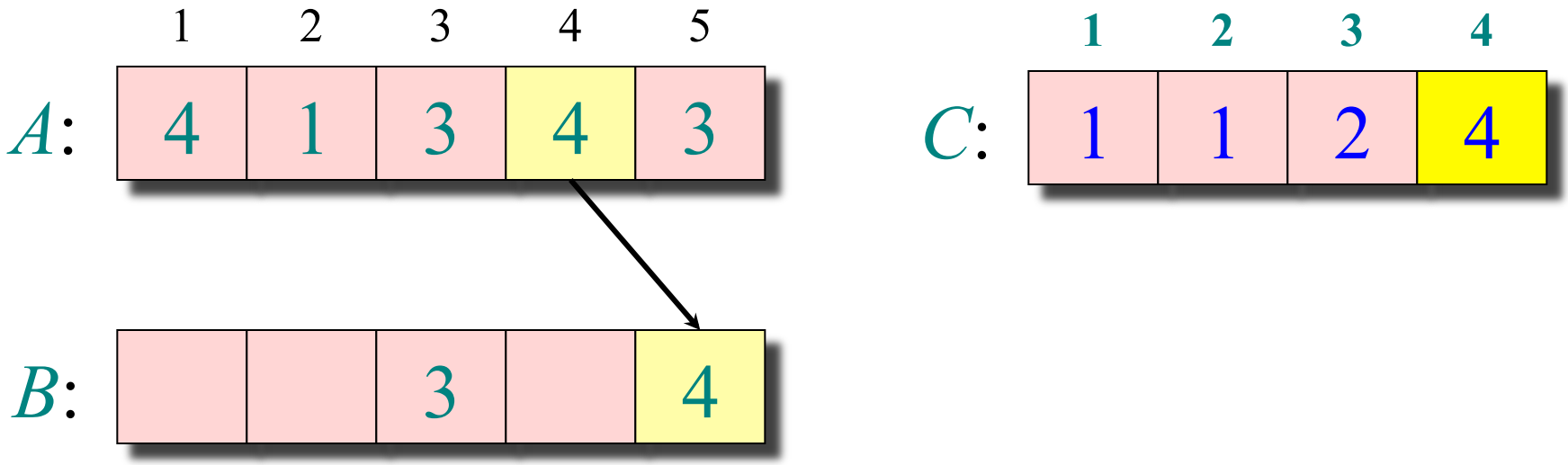




There are exactly 5 elements ≤A[4], so where should I place A[4]?

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

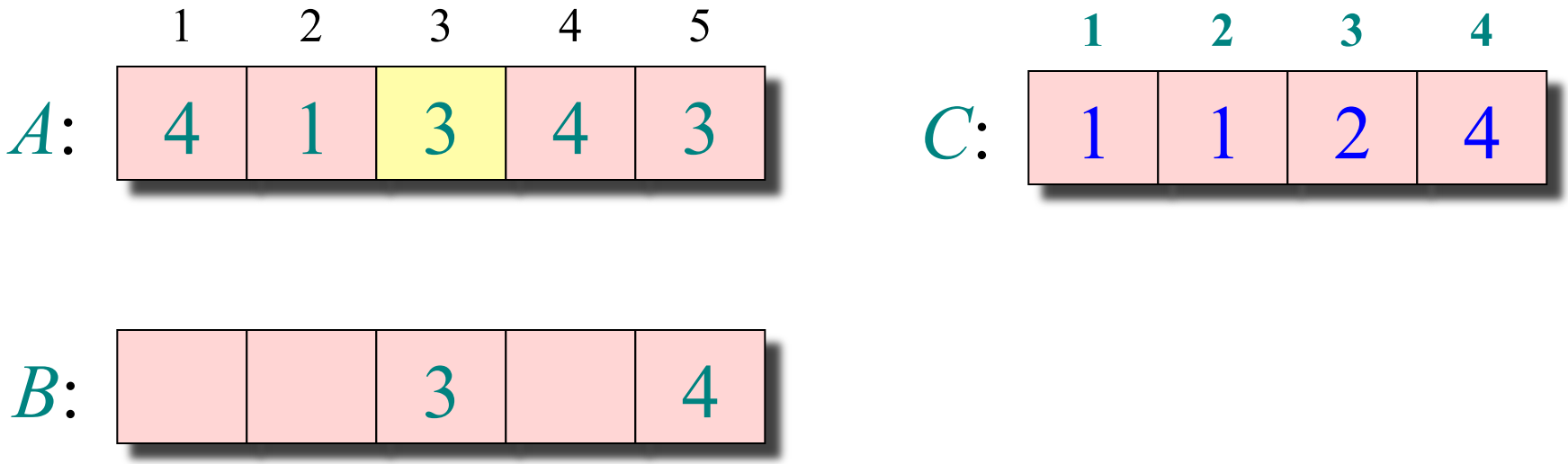


for $j \leftarrow n$ **downto** 1

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

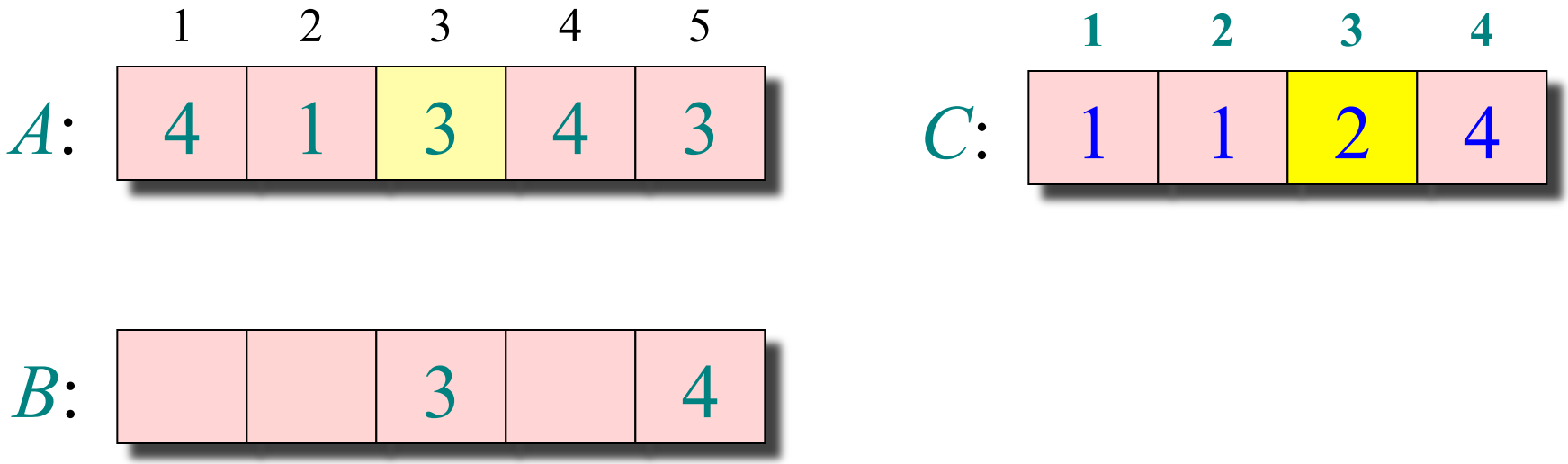
$C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A



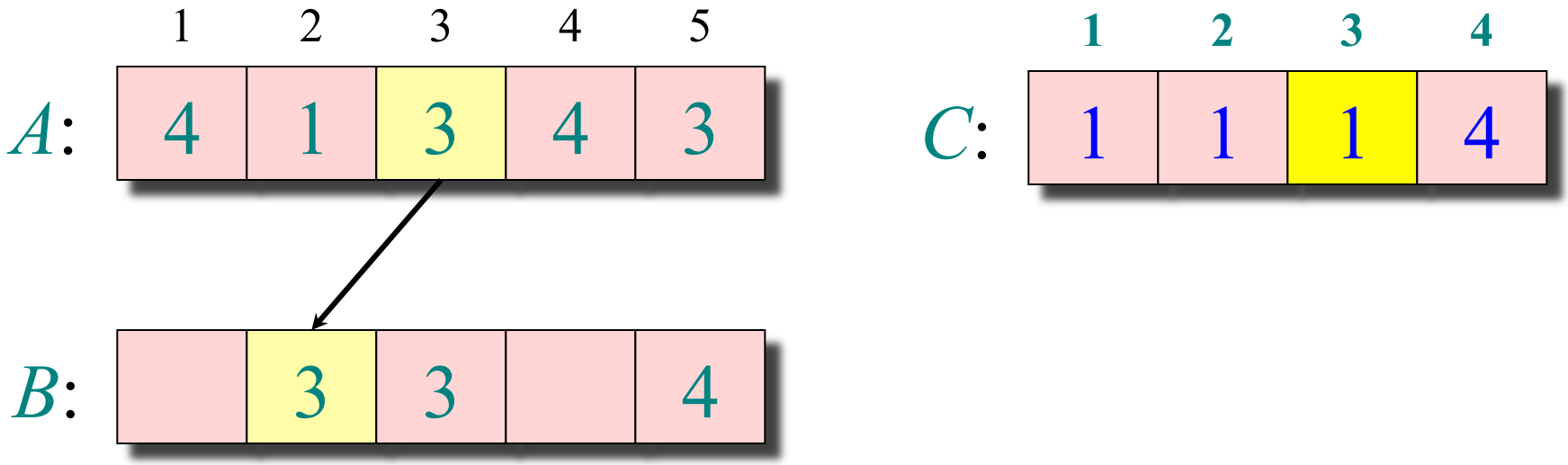
for $j \leftarrow n$ **downto** 1
    do $B[C[A[j]]] \leftarrow A[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$

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

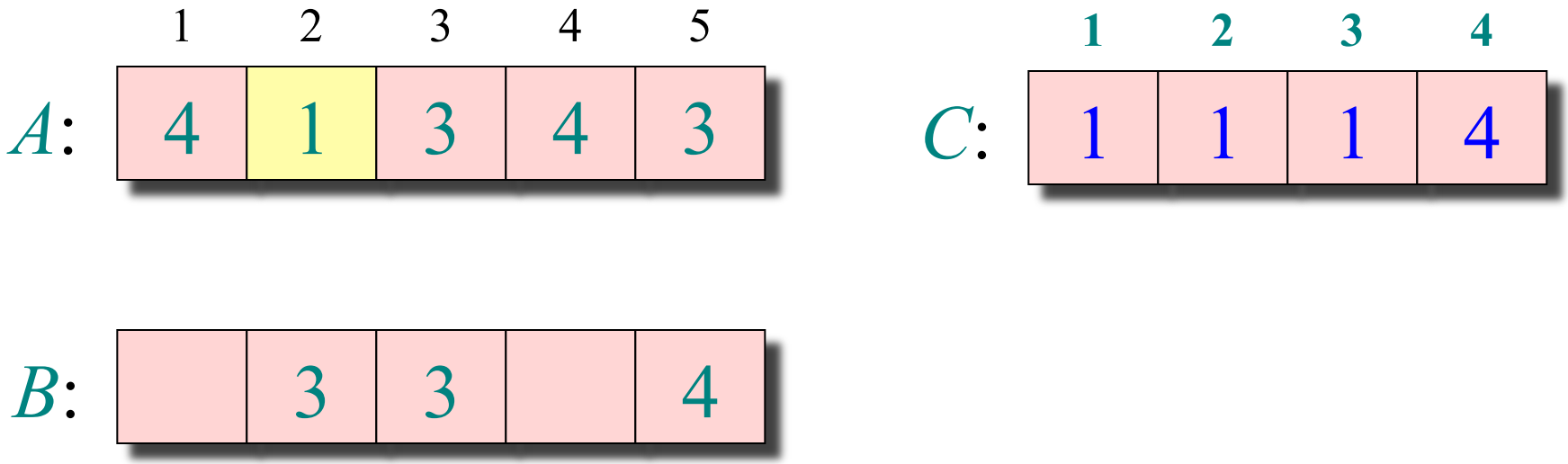


for $j \leftarrow n$ **downto** 1

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

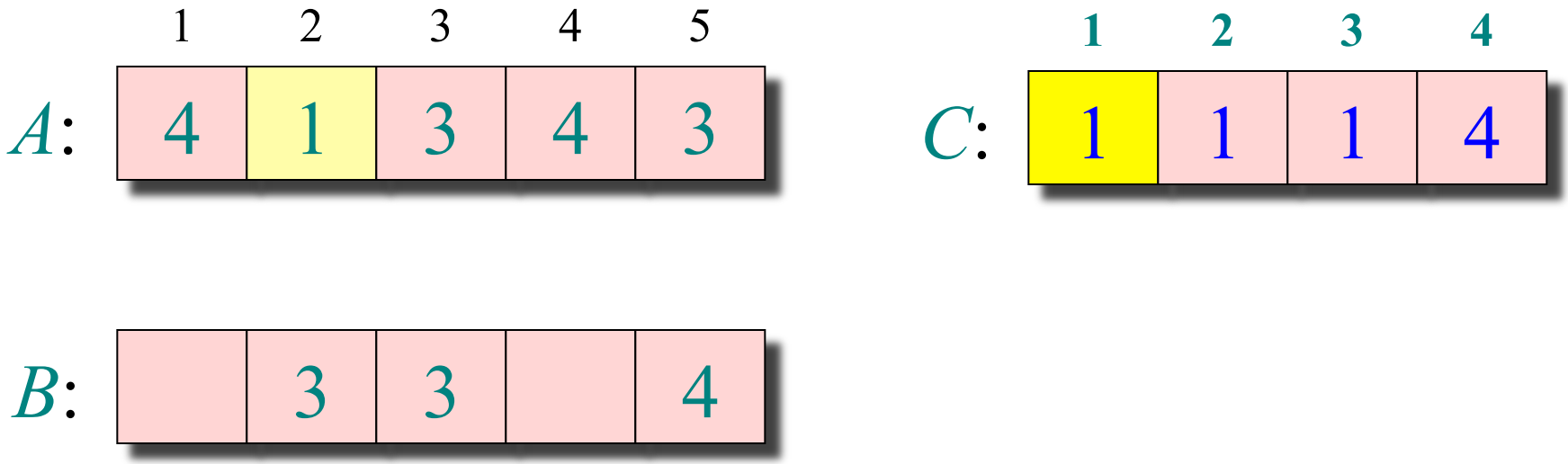
$C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A



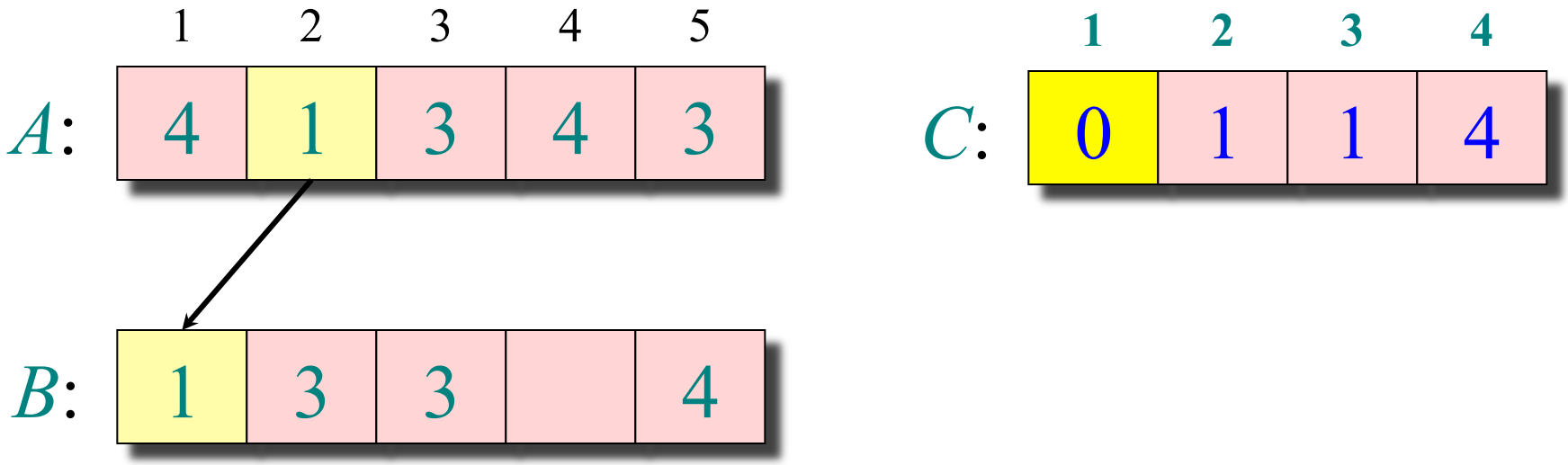
for $j \leftarrow n$ **downto** 1
    do $B[C[A[j]]] \leftarrow A[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$

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

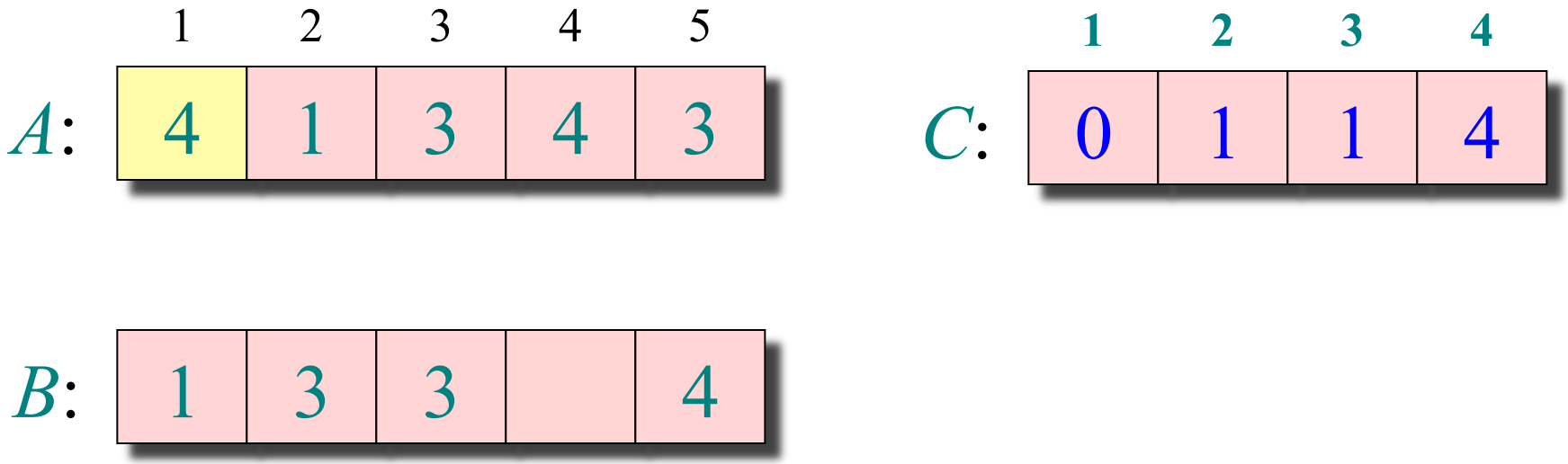


for $j \leftarrow n$ **downto** 1

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

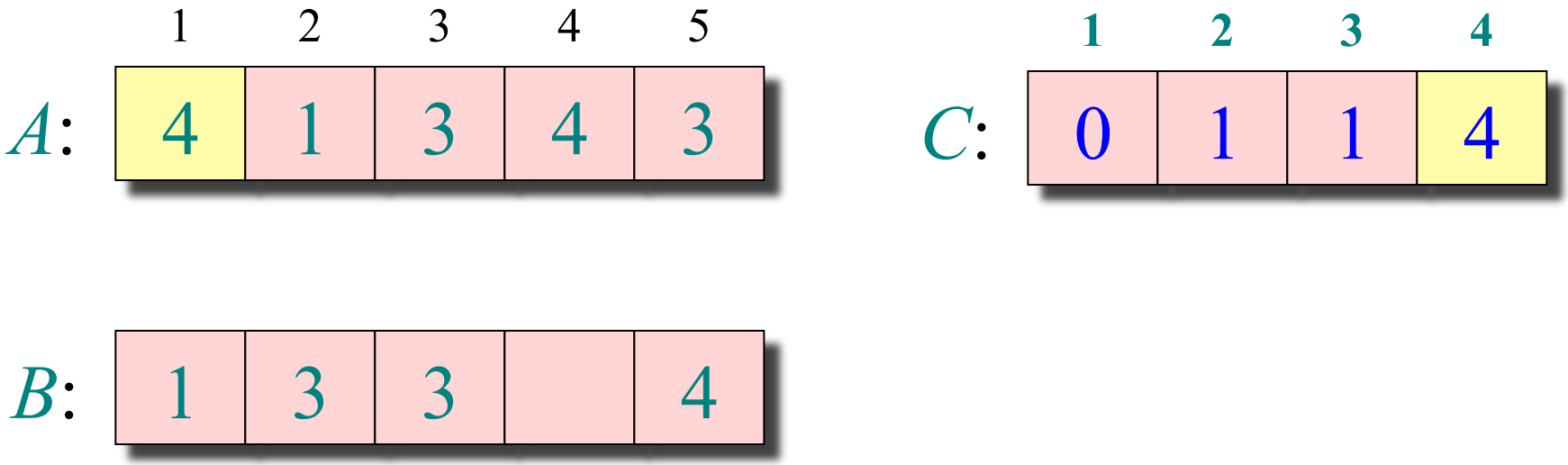
$C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4: permute elements of A



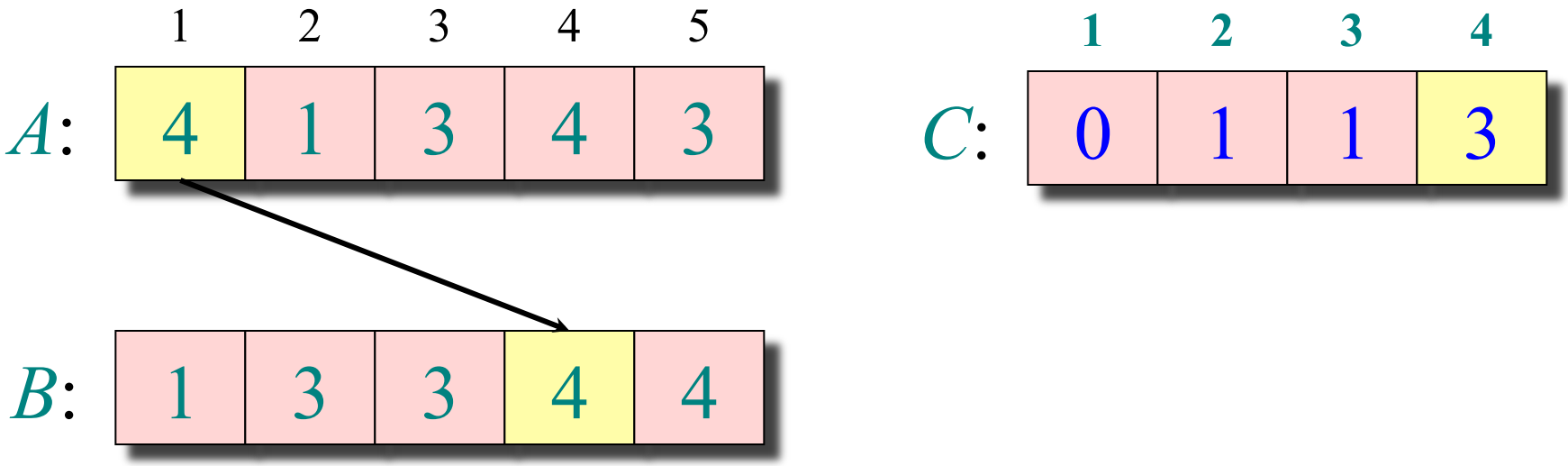
for $j \leftarrow n$ **downto** 1
    do $B[C[A[j]]] \leftarrow A[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$

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

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

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.

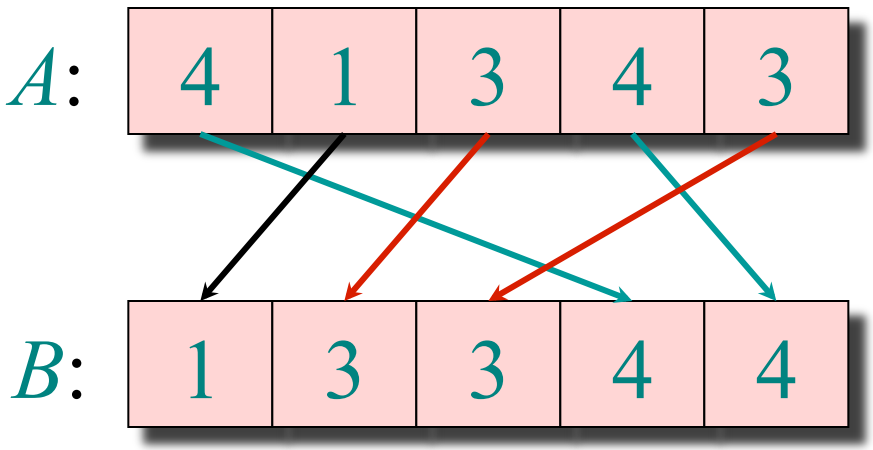
- But, sorting takes $\Omega(n \lg n)$ time!
- Where's the fallacy?

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

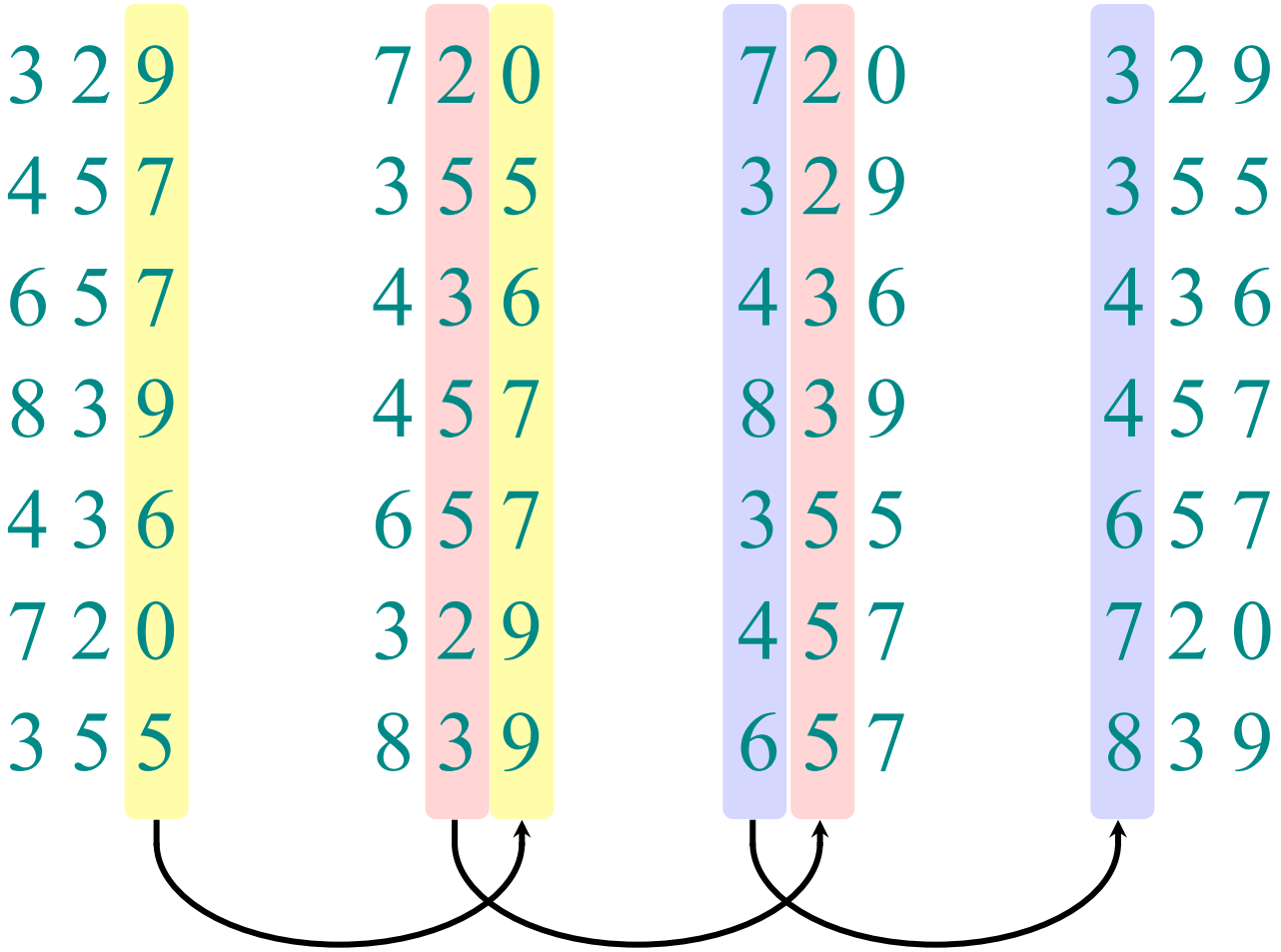
Counting sort is a ***stable*** sort: it preserves the input order among equal elements.



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

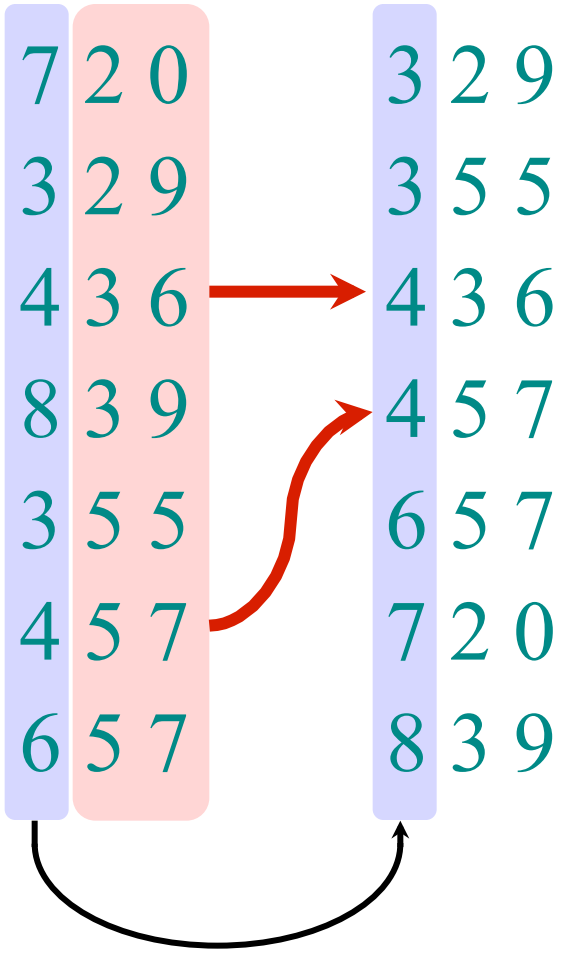


Additional material
(not covered in Quiz 1)

Correctness of radix sort

Induction on digit position

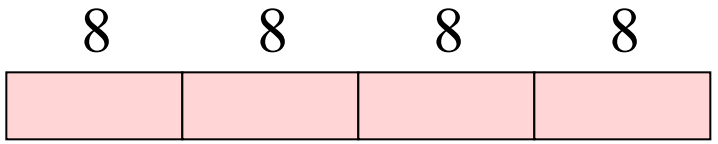
- Assume that the numbers are sorted by their low-order $t - 1$ digits.
- Sort on digit t
  - Two numbers that differ in digit t are correctly sorted.
  - Two numbers equal in digit t are put in the same order as the input $\Rightarrow$ correct order.



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

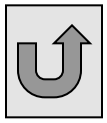


- If each b-bit word is broken into r-bit pieces, each pass of counting sort takes $\Theta(n + 2^r)$ time.
- Setting $r = \log n$ gives $\Theta(n)$ time per pass, or $\Theta(n\, b/\log n)$ total

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.

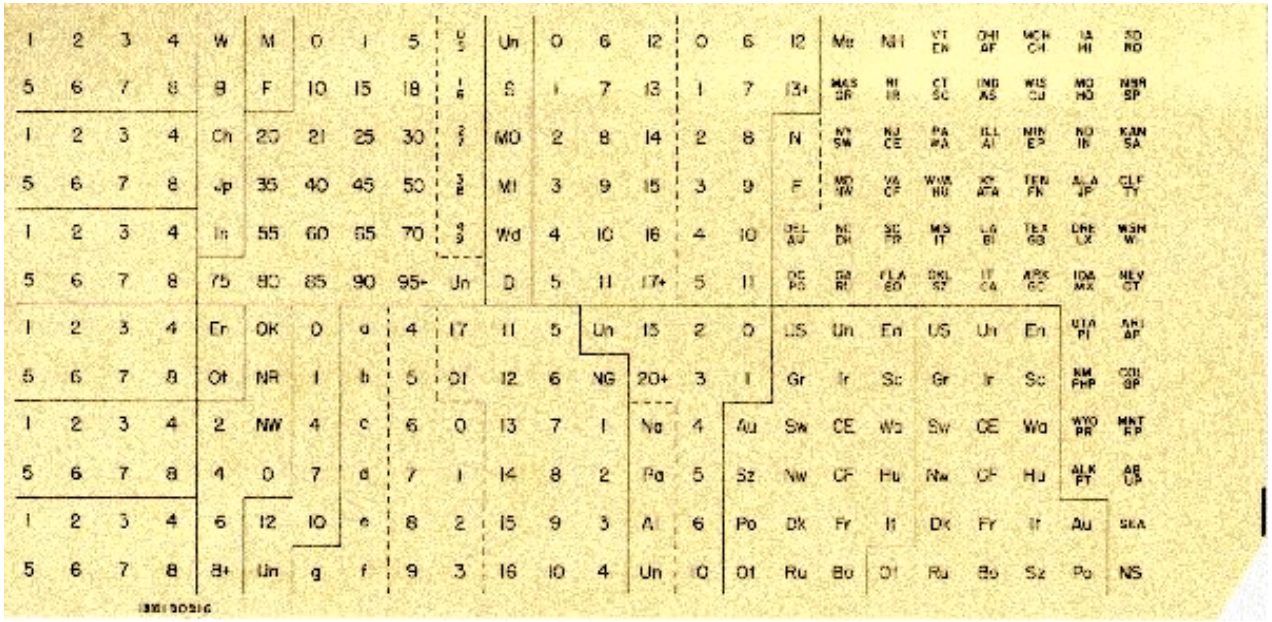


Herman Hollerith (1860-1929)

- 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

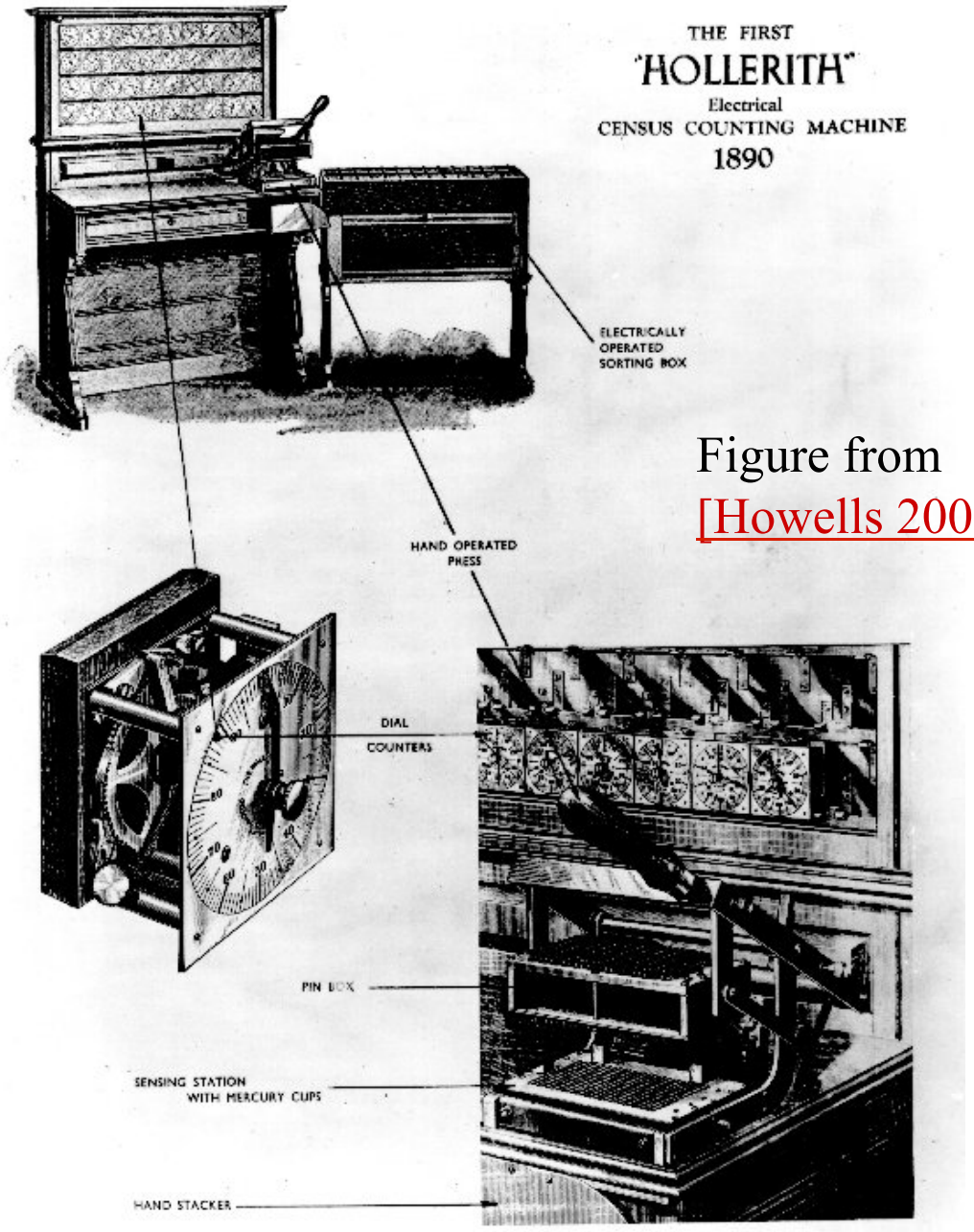
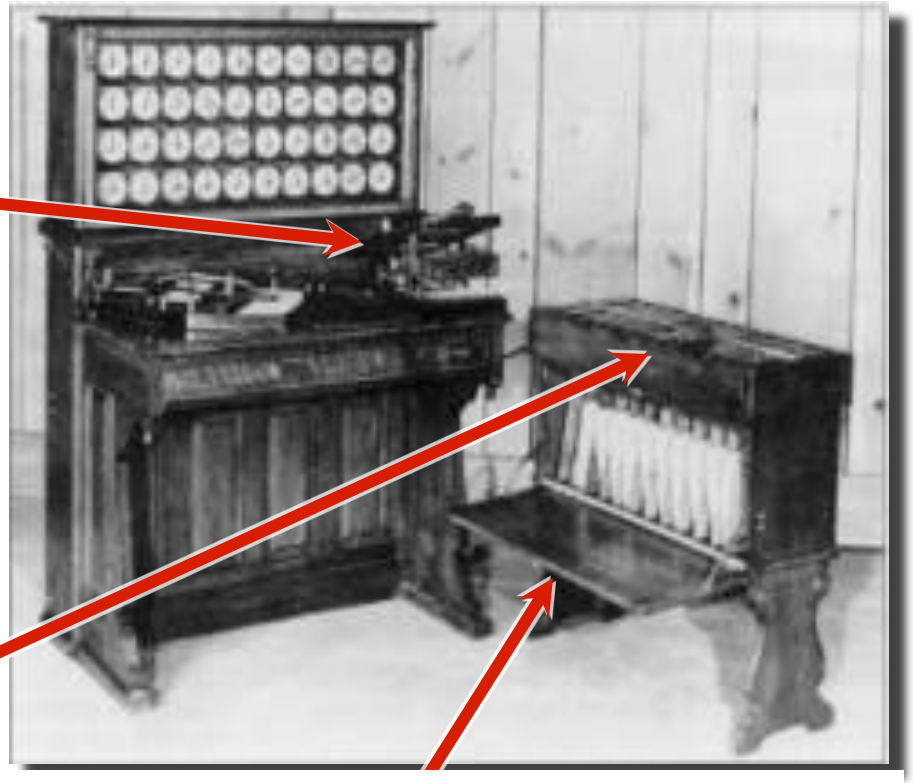


Figure from [Howells 2000].

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.



Hollerith Tabulator, Pantograph, Press, and Sorter

- 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.

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!

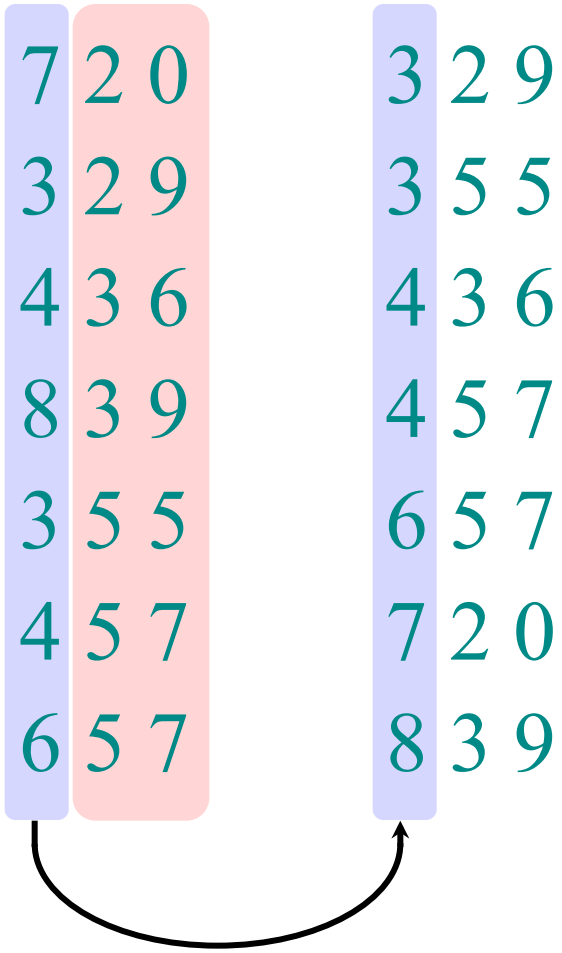
Web resources on punched-card technology

- 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

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.