数据结构与算法设计

周　可

Mail : zhke@hust.edu.cn

华中科技大学，武汉光电国家研究中心

Quicksort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts "in place" (like insertion sort, but not like merge sort).
- Very practical (with tuning).
- Quicksort is typically over twice as fast as merge sort.

快速排序是一种基于划分的排序方法。

划分：在待分类集合A中选取某元素t，按照与t的大小关系重新整理A中元素，使得整理后t被置于序列的某位置上，而在t以前出现的元素均小于等于t，在t以后的元素均大于等于t。这一元素的整理过程称为划分（Partitioning）。

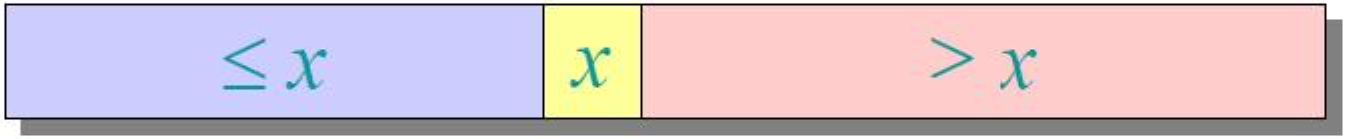
划分元素：元素t被称为划分元素（有的文献称为pivot，轴元素）。

快速排序：这种通过对待排序集合反复划分达到排序目的的算法称为快速排序算法。

Divide and conquer

Quicksort an n-element array:

1. ***Divide:*** Partition the array into two subarrays around a ***pivot*** x such that elements in lower subarray $\leq x <$ elements in upper subarray.



2. ***Conquer:*** Recursively sort the two subarrays.
3. ***Combine:*** Trivial.

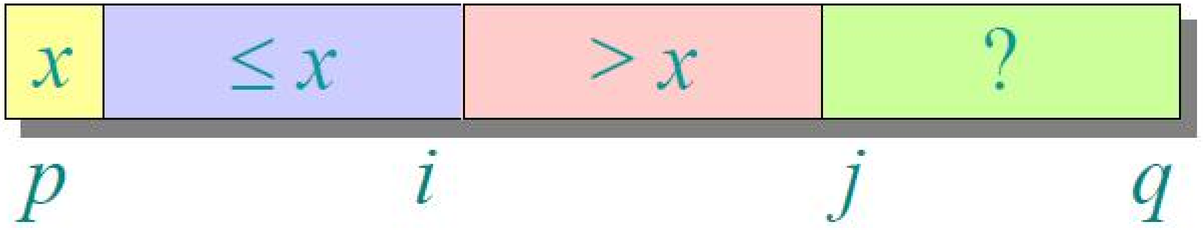
Key: *Linear-time partitioning subroutine.*

Partitioning subroutine

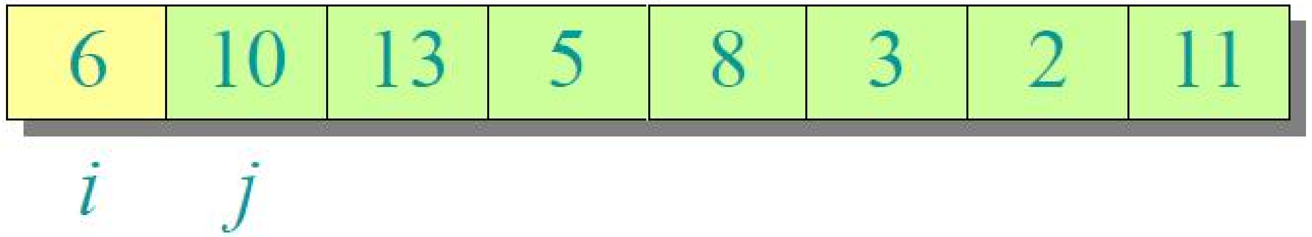
```
PARTITION($A, p, q$)   ▷ $A[p\,.\,.\,q]$
    $x \leftarrow A[p]$          ▷ pivot = $A[p]$
    $i \leftarrow p$
    for $j \leftarrow p + 1$ to $q$
        do if $A[j] \le x$
              then $i \leftarrow i + 1$
                   exchange $A[i] \leftrightarrow A[j]$
    exchange $A[p] \leftrightarrow A[i]$
    return $i$
```

Running time $= O(n)$ for n elements.

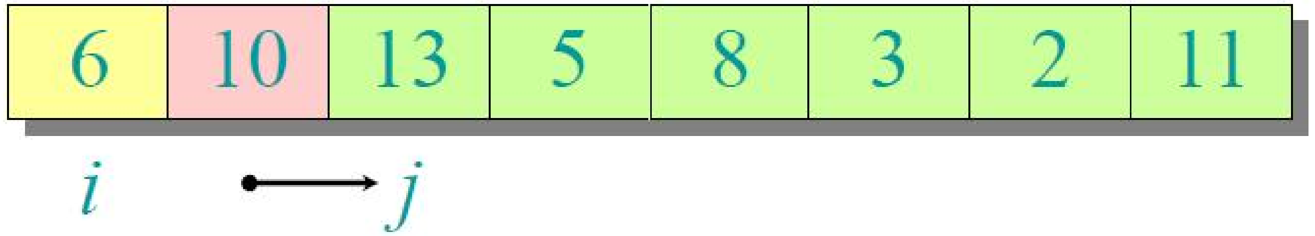
Invariant:



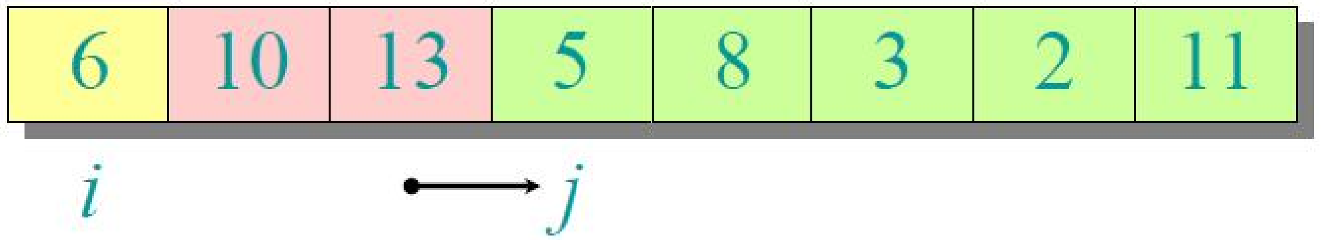
Example of partitioning



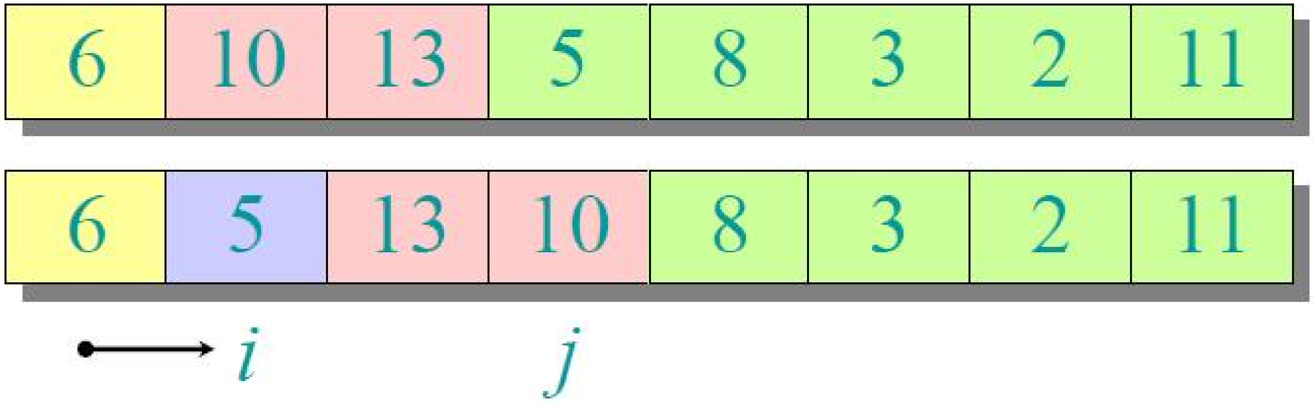
Example of partitioning



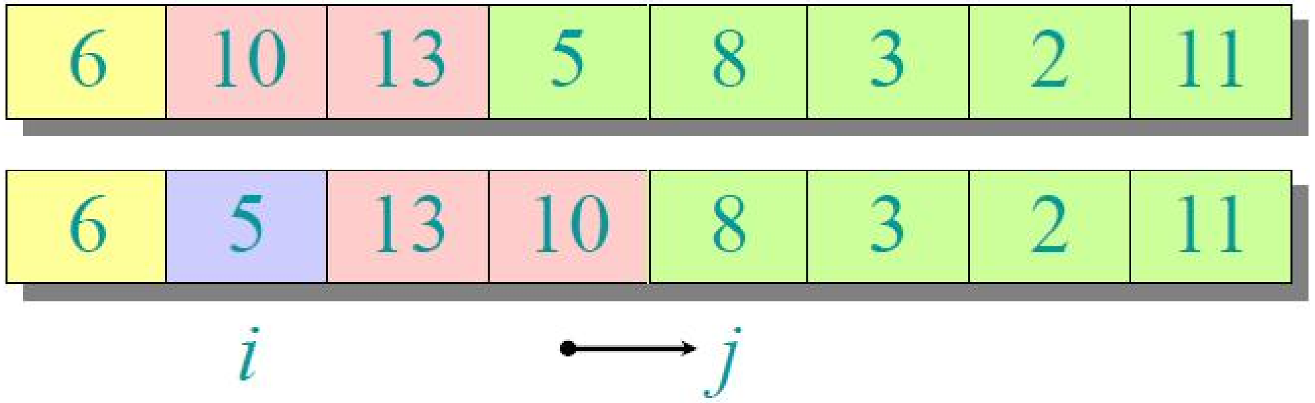
Example of partitioning



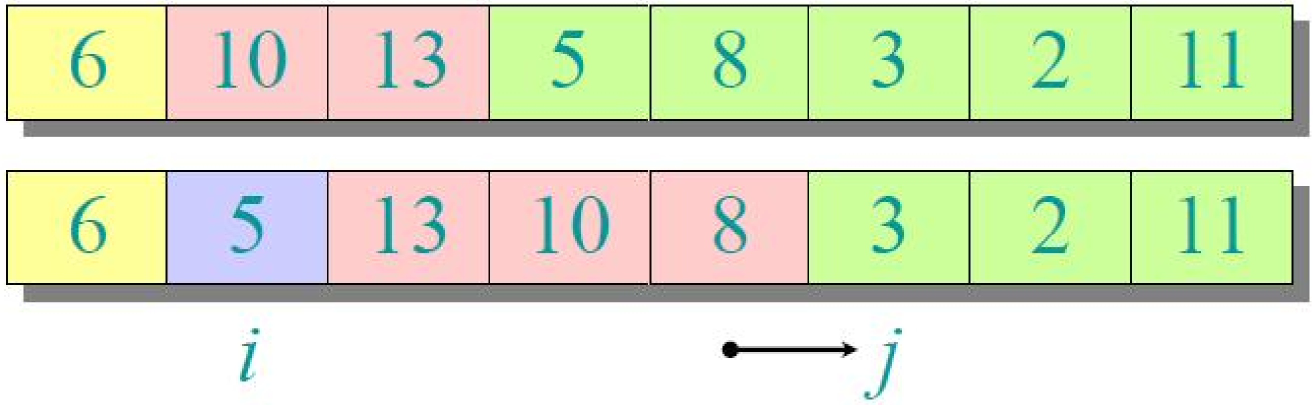
Example of partitioning



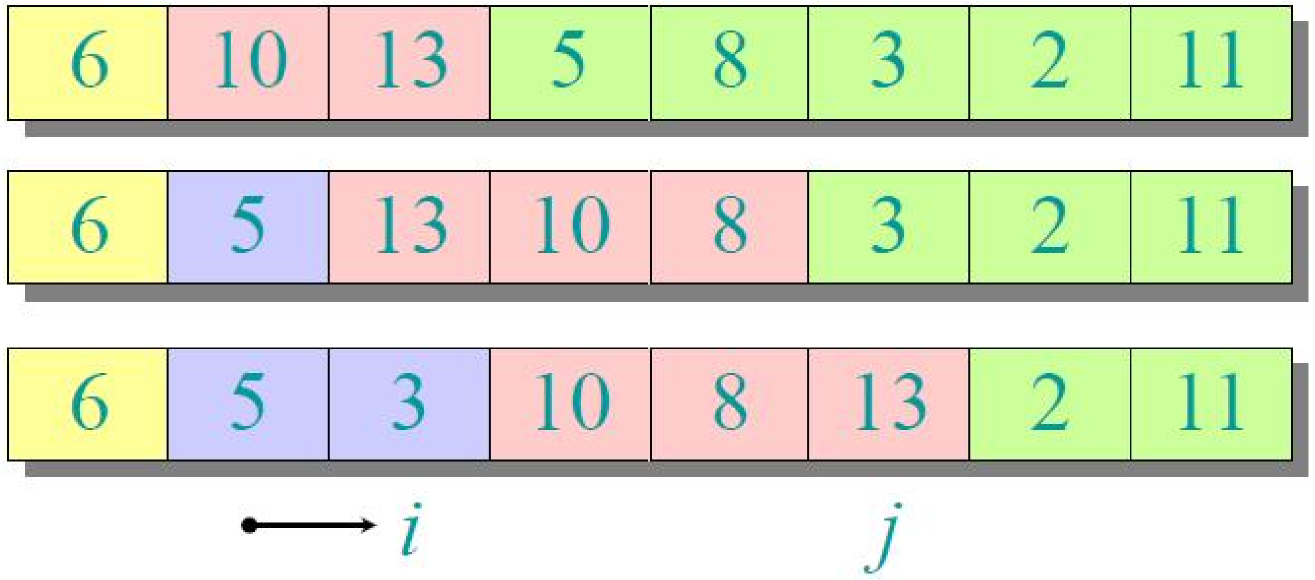
Example of partitioning



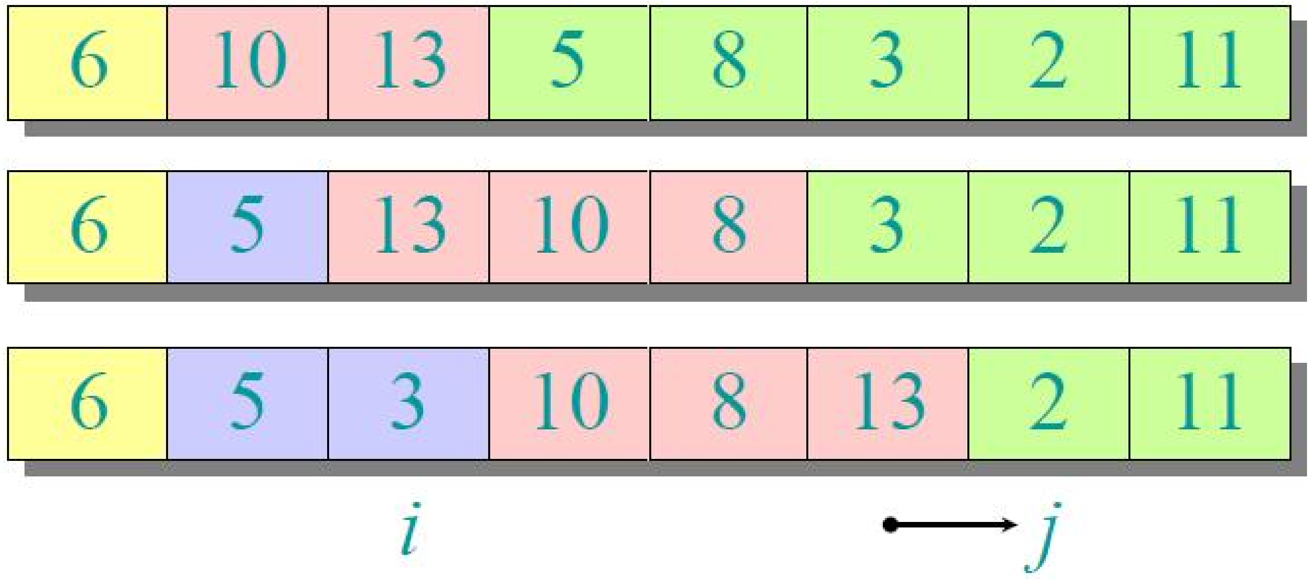
Example of partitioning



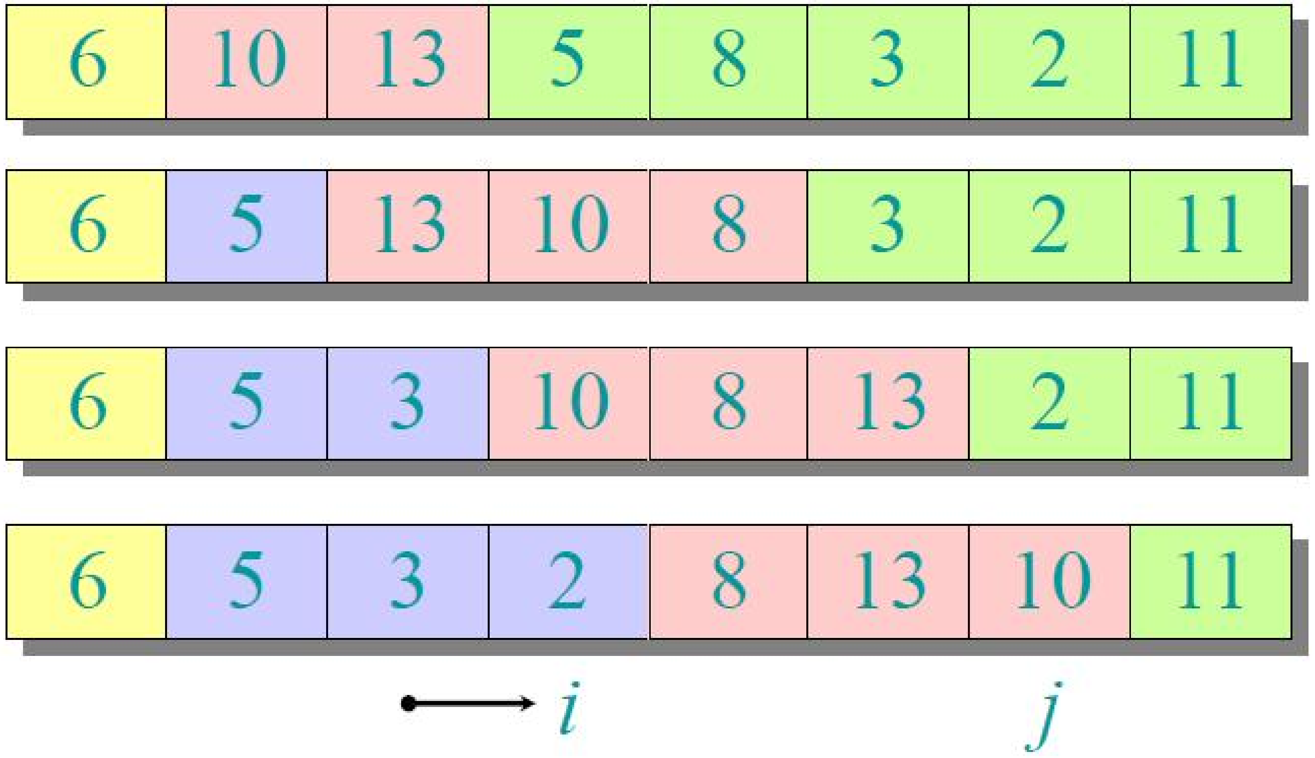
Example of partitioning



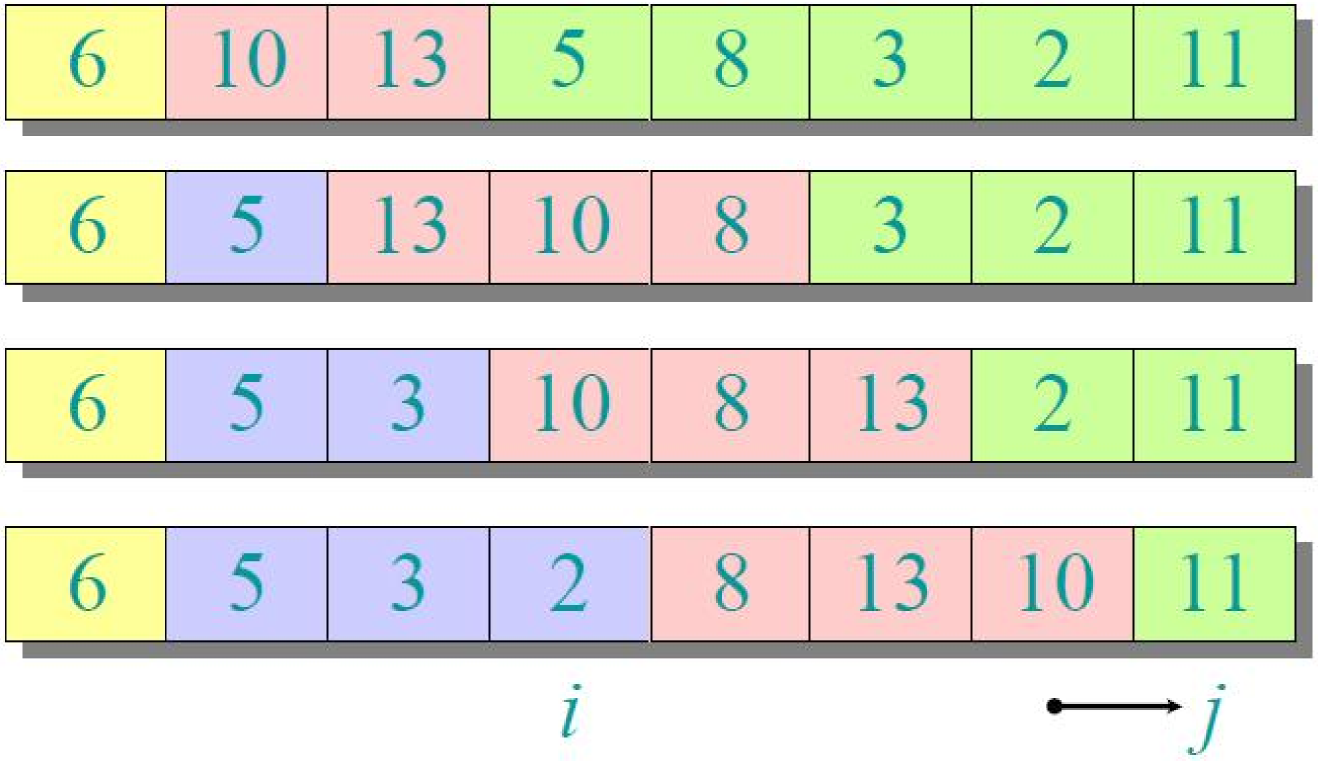
Example of partitioning



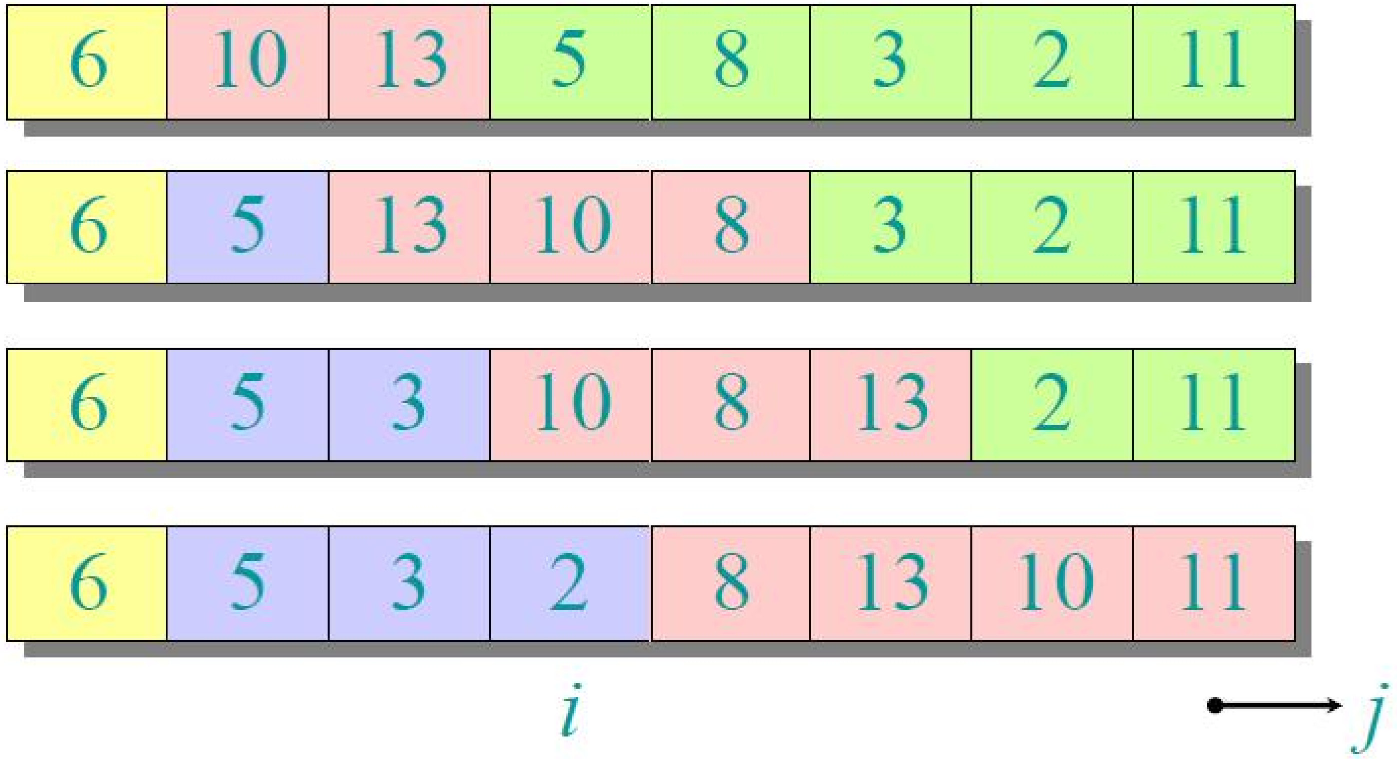
Example of partitioning



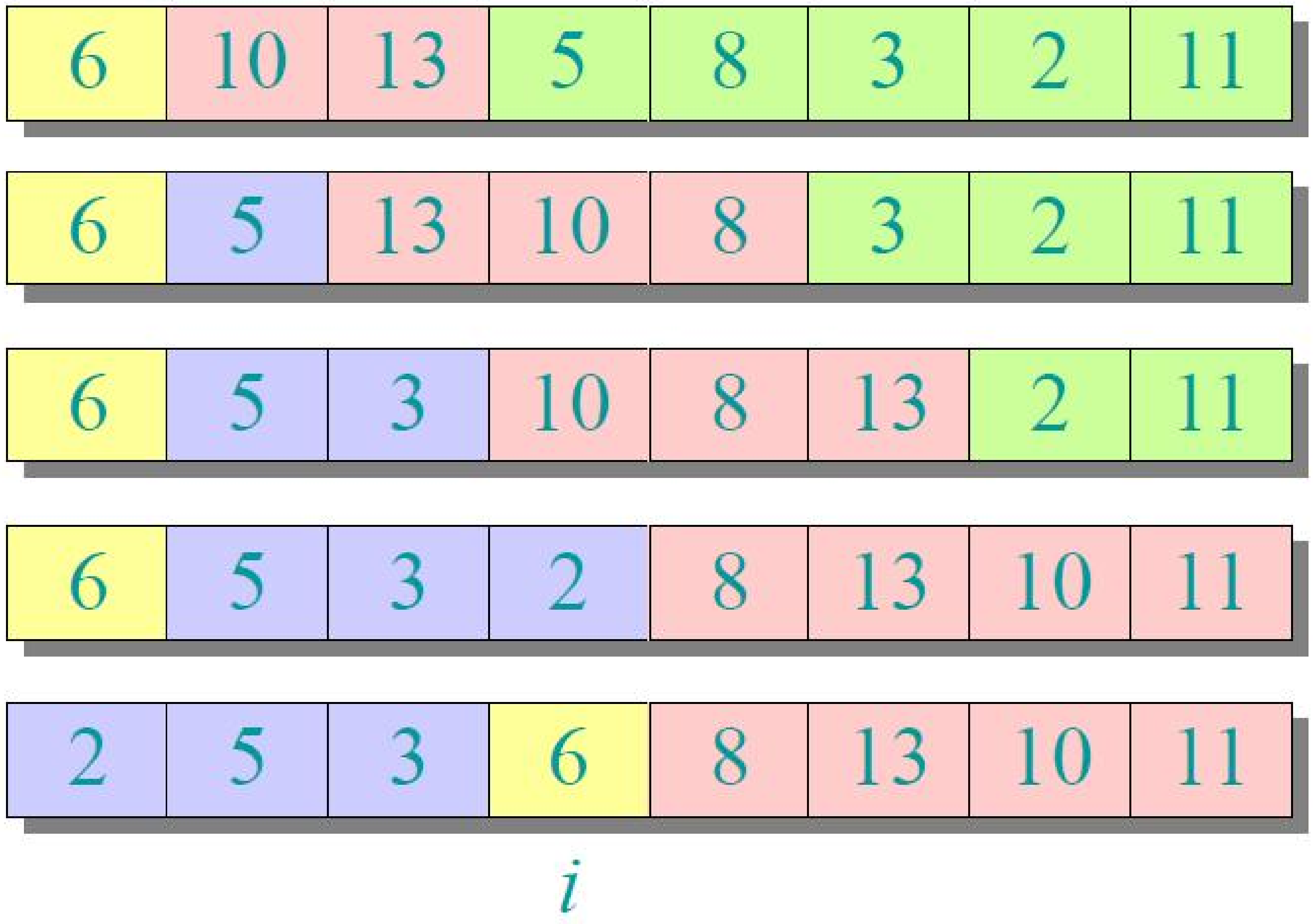
Example of partitioning



Example of partitioning



Example of partitioning



Pseudocode for quicksort

```
QUICKSORT($A$, $p$, $r$)
    if $p < r$
        then $q \leftarrow$ PARTITION($A$, $p$, $r$)
            QUICKSORT($A$, $p$, $q-1$)
            QUICKSORT($A$, $q+1$, $r$)
```

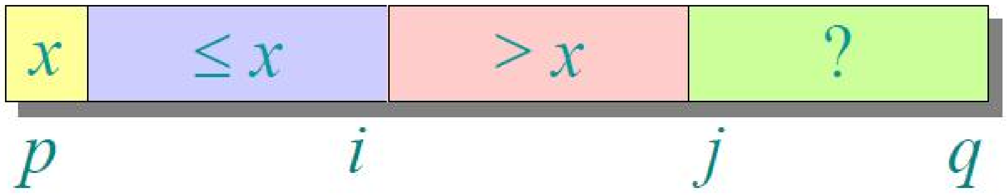
Initial call: QUICKSORT(A, 1, n)

Partitioning subroutine

```
PARTITION($A, p, q$)  ▷ $A[p\,..\,q]$
    $x \leftarrow A[p]$          ▷ pivot $= A[p]$
    $i \leftarrow p$
    for $j \leftarrow p + 1$ to $q$
        do if $A[j] \leq x$
              then  $i \leftarrow i + 1$
                    exchange $A[i] \leftrightarrow A[j]$
    exchange $A[p] \leftrightarrow A[i]$
    return $i$
```

Running time $= O(n)$ for n elements.

Invariant:



Analysis of quicksort

- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms for when duplicate input elements may exist.
- Let $T(n)$ = worst-case running time on an array of n elements.

Worst-case of quicksort

- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

$$\begin{aligned} T(n) &= T(0) + T(n-1) + \Theta(n) \\ &= \Theta(1) + T(n-1) + \Theta(n) \\ &= T(n-1) + \Theta(n) \\ &= \Theta(n^2) \qquad \textit{(arithmetic series)} \end{aligned}$$

Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

$T(n)$

Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

$$\begin{tikzcd}
& cn \arrow[dl, no head] \arrow[dr, no head] & \\
T(0) & & T(n-1)
\end{tikzcd}$$

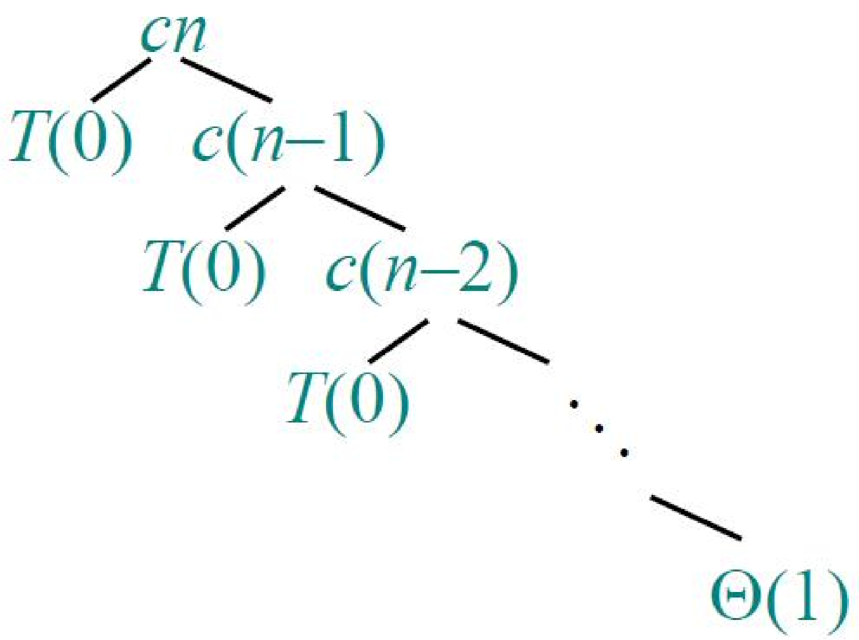
Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

$$\begin{tikzcd}[arrows=dash]
 & cn \arrow[dl] \arrow[dr] & & \\
T(0) & & c(n-1) \arrow[dl] \arrow[dr] & \\
 & T(0) & & T(n-2)
\end{tikzcd}$$

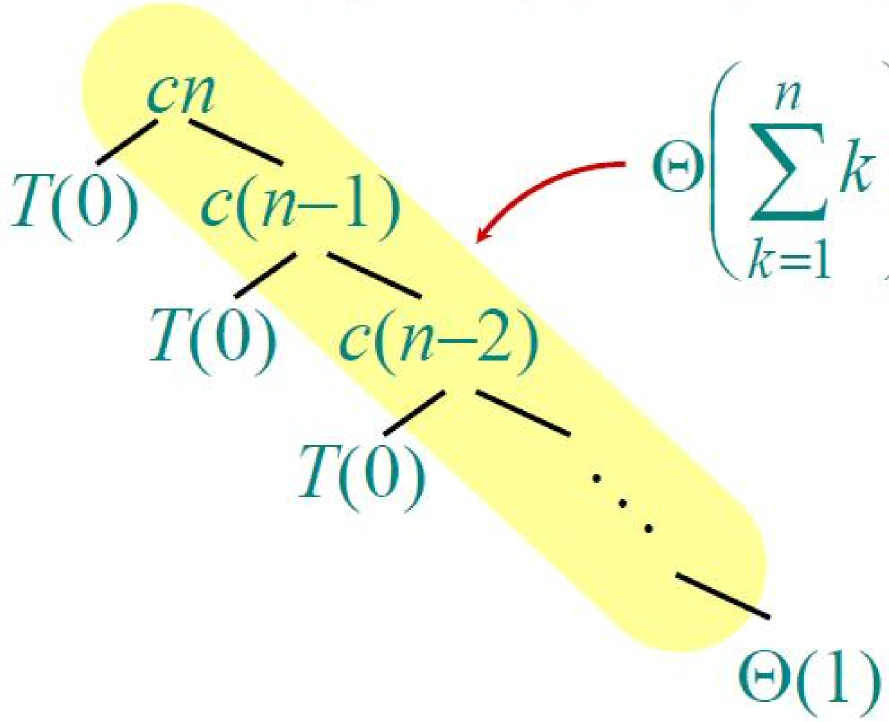
Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$



Worst-case recursion tree

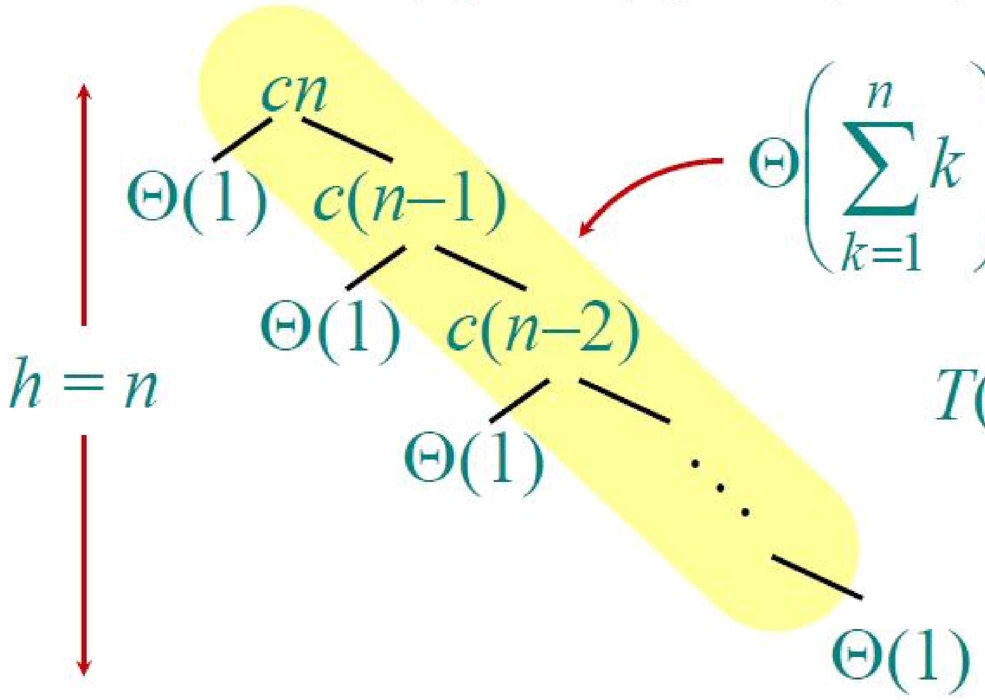
$$T(n) = T(0) + T(n-1) + cn$$



$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$

Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$



$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$

$$T(n) = \Theta(n) + \Theta(n^2)$$
$$= \Theta(n^2)$$

Best-case analysis

(For intuition only!)

If we're lucky, PARTITION splits the array evenly:

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

(same as merge sort)

What if the split is always $\frac{1}{10} : \frac{9}{10}$?

$$T(n) = T\left(\tfrac{1}{10}n\right) + T\left(\tfrac{9}{10}n\right) + \Theta(n)$$

What is the solution to this recurrence?

Analysis of “almost-best” case

$T(n)$

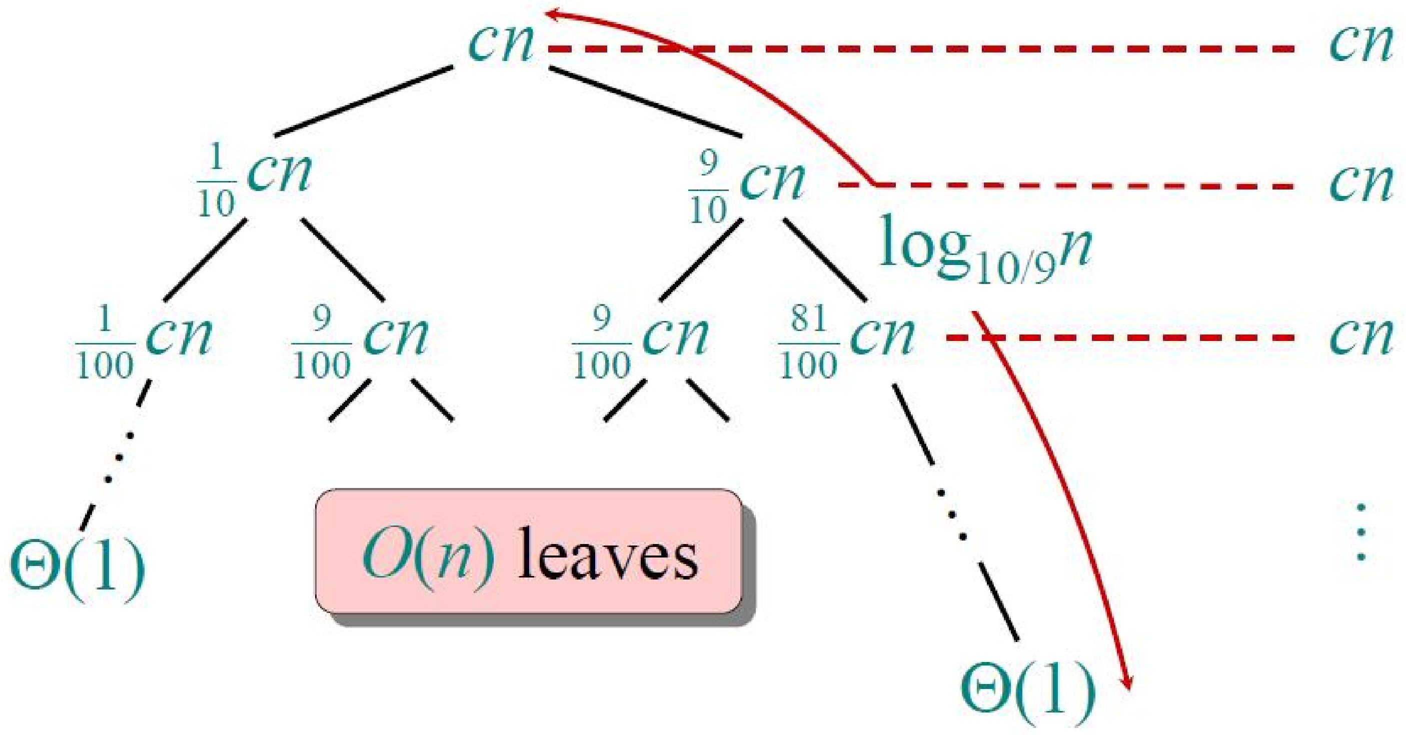
Analysis of “almost-best” case

$$\begin{tikzcd}
 & cn \arrow[dl, no head] \arrow[dr, no head] & \\
T\left(\frac{1}{10}n\right) & & T\left(\frac{9}{10}n\right)
\end{tikzcd}$$

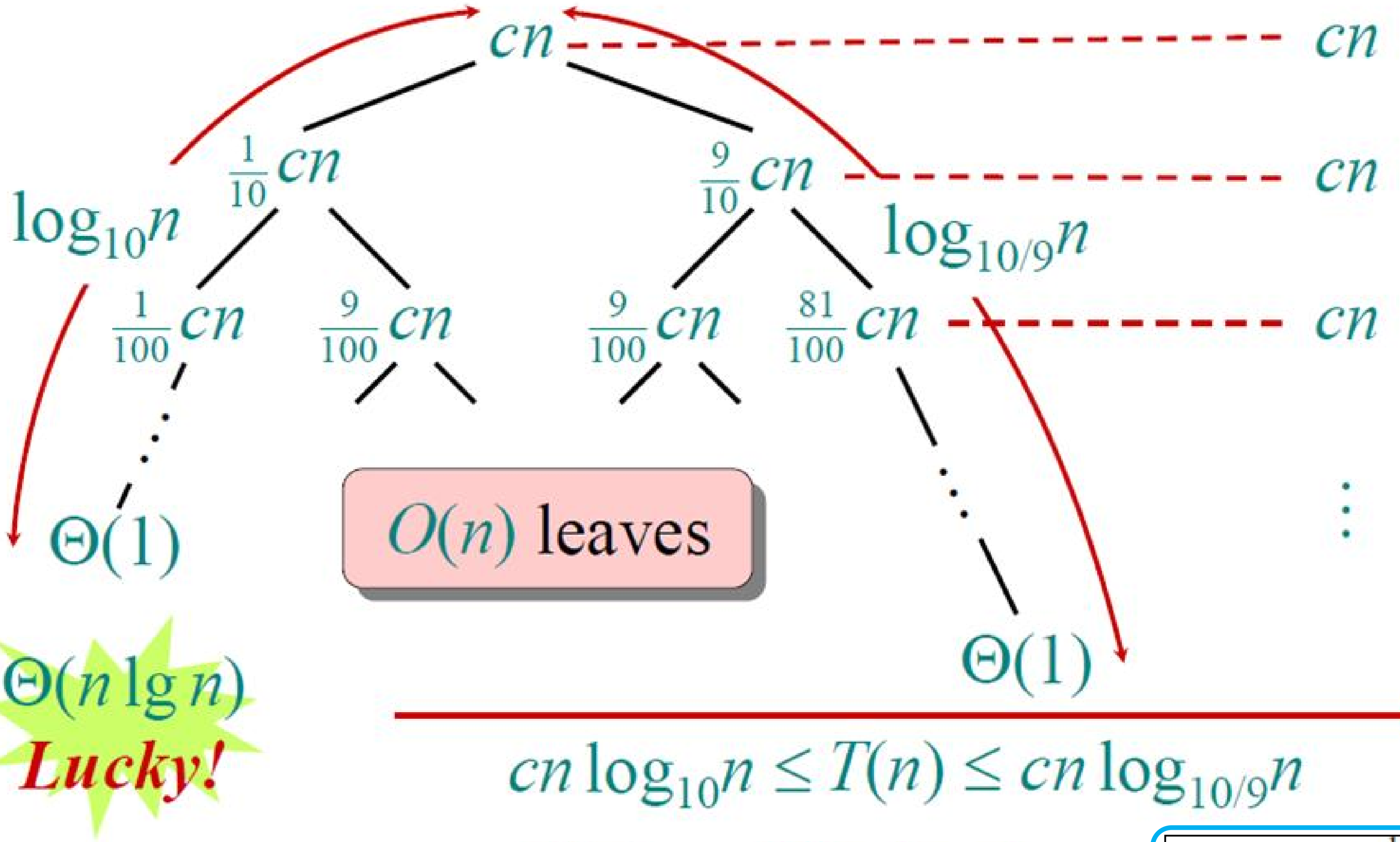
Analysis of “almost-best” case

$$\begin{tikzcd}[column sep=tiny]
 & & & cn \arrow[dll, no head] \arrow[drr, no head] & & & \\
 & \frac{1}{10}cn \arrow[dl, no head] \arrow[dr, no head] & & & & \frac{9}{10}cn \arrow[dl, no head] \arrow[dr, no head] & \\
T\left(\frac{1}{100}n\right) & & T\left(\frac{9}{100}n\right) & & T\left(\frac{9}{100}n\right) & & T\left(\frac{81}{100}n\right)
\end{tikzcd}$$

Analysis of “almost-best” case



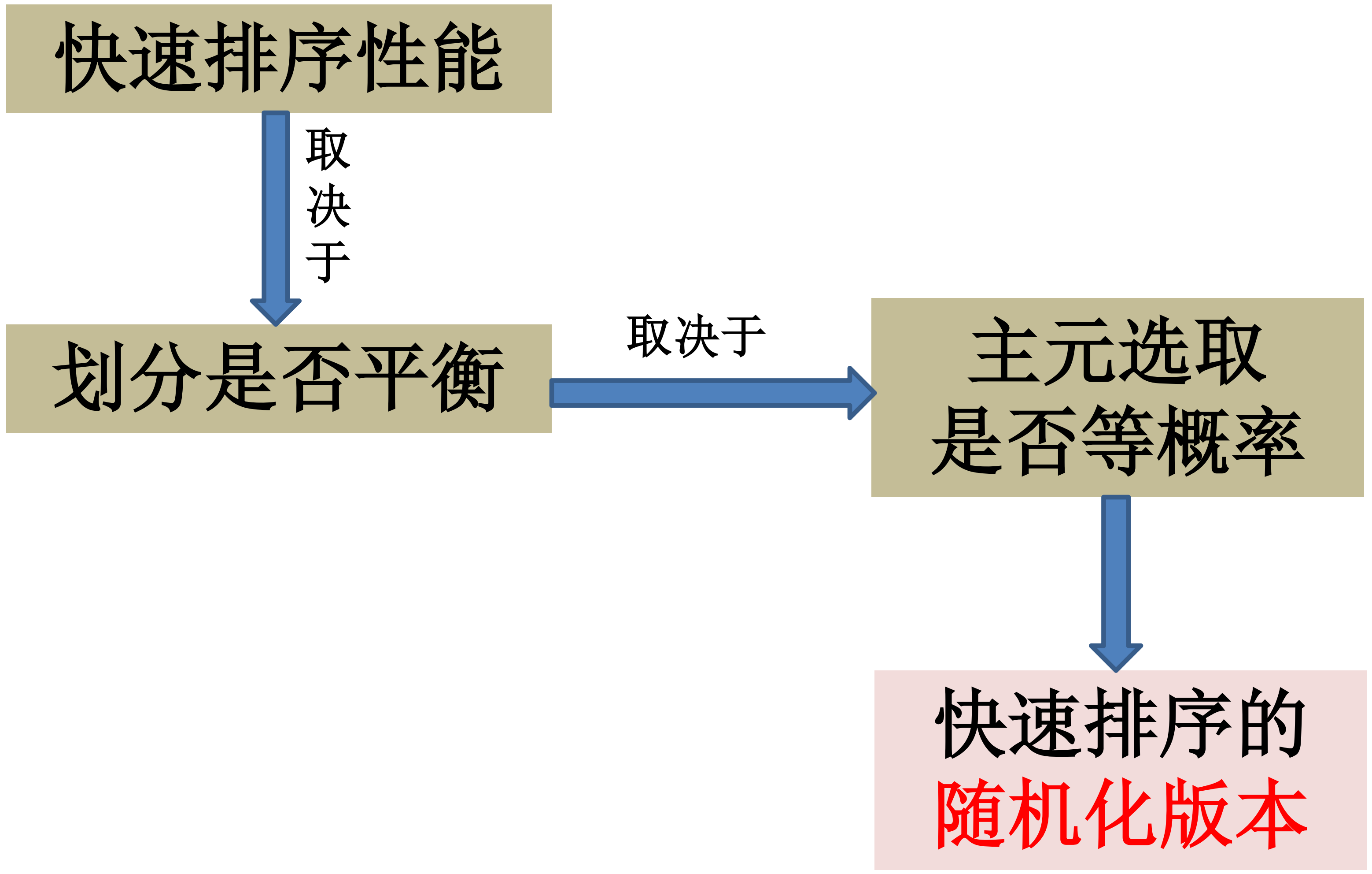
Analysis of "almost-best" case



$$cn \log_{10} n \le T(n) \le cn \log_{10/9} n$$

$$T(n) = \Theta(n \lg n)$$

$$\boxed{\log_b a = \frac{\log_c a}{\log_c b}}$$


快速排序性能
取决于
划分是否平衡
取决于
主元选取是否等概率
快速排序的随机化版本

Randomized quicksort

IDEA: Partition around a ***random*** element.

- Running time is independent of the input order.
- No assumptions need to be made about the input distribution.
- No specific input elicits the worst-case behavior.
- The worst case is determined only by the output of a random-number generator.

Randomized quicksort

```
Randomized-quicksort(A, p, r)
If p<r
  then q ← Randomized-partition(A, p, r)
        Randomized-quicksort(A, p, q-1)
        Randomized-quicksort(A, q+1, r)

Randomized-partition(A, p, r)
   i ← Random(p, r)
   exchange A[p] with A[i]
   return partition(A, p, r)
```

About QuickSort:

The quicksort algorithm has a worst-case running time of $O(n^2)$ on an input array of n numbers. Despite this slow worst-case running time, quicksort is often the best practical choice for sorting because it is remarkably efficient on the average: its expected running time is $\Theta(n\lg n)$, and the constant factors hidden in the $\Theta(n\lg n)$ notation are quite small.

How fast can we 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, quicksort, heapsort.

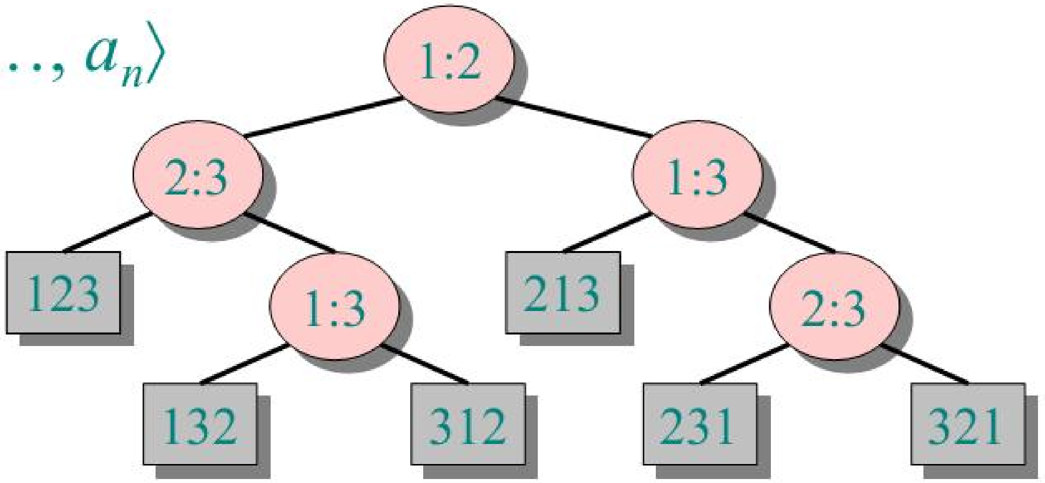
The best worst-case running time that we've seen for comparison sorting is $O(n \lg n)$.

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

Decision trees can help us answer this question.

Decision-tree example

Sort $\langle a_1, a_2, \ldots, a_n \rangle$

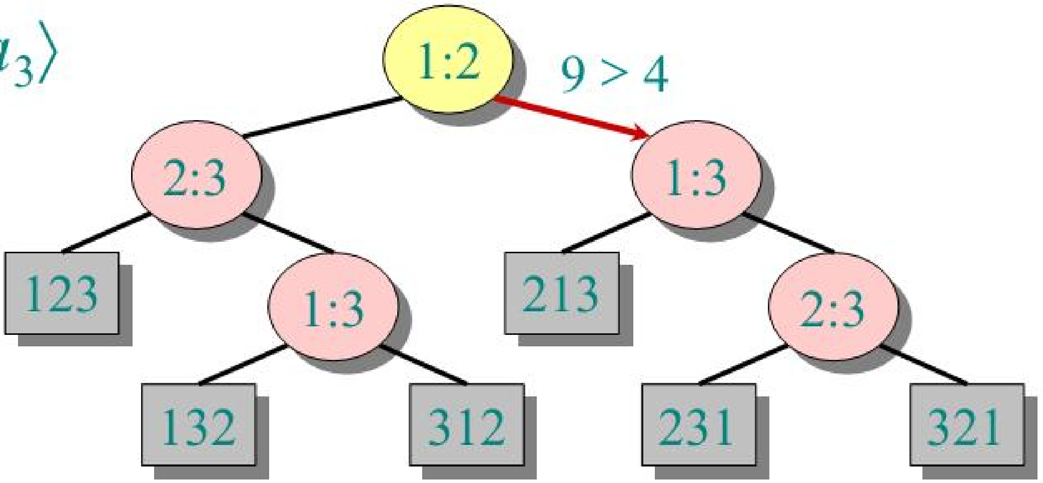


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

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

Decision-tree example

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

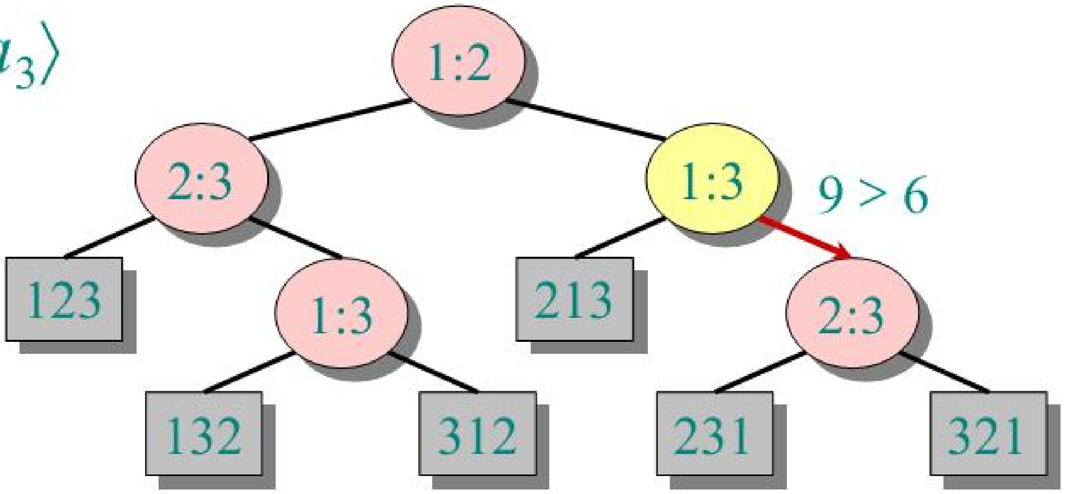


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

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

Decision-tree example

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

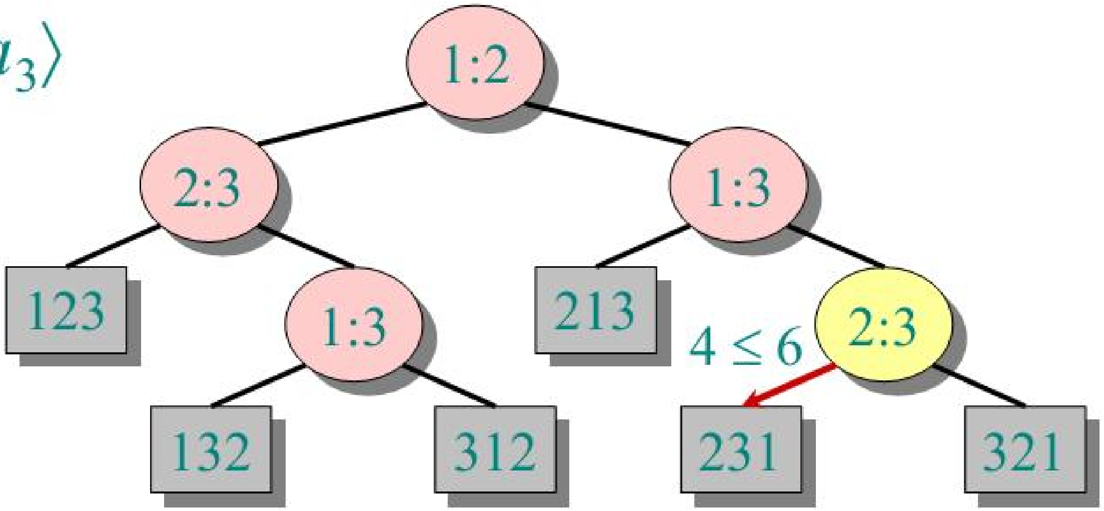


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

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

Decision-tree example

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

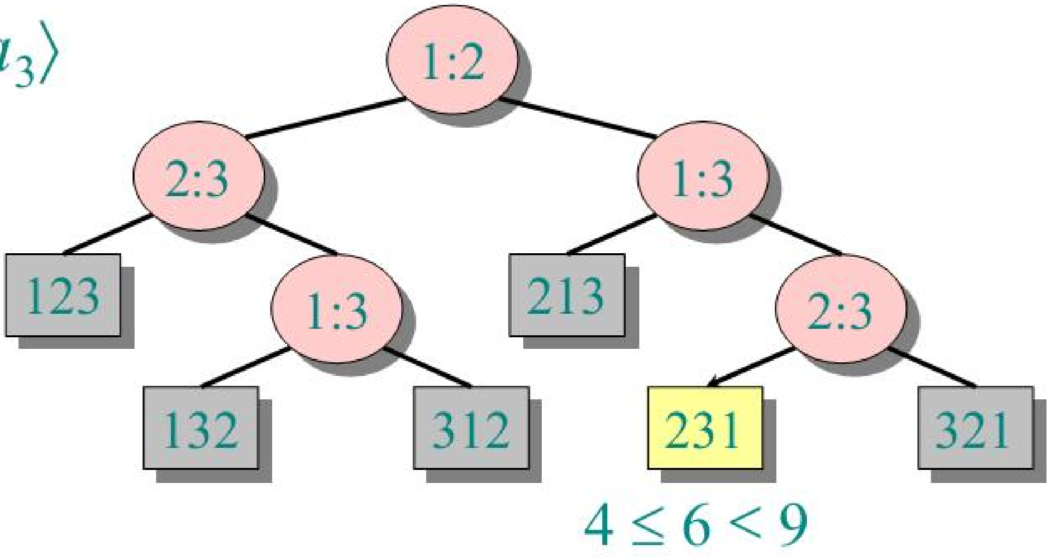


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \dots, n\}$.

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

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.
- View the algorithm as splitting whenever it compares two elements.
- The tree contains the comparisons along all possible instruction traces.
- 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, $n! \leq 2^h$.

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

Lower bound for comparison sorting

Corollary. Heapsort and merge sort are asymptotically optimal comparison sorting algorithms.

4.1 Quicksort

4.2 Decision tree

4.3 Counting sort

Sorting in linear time

Counting sort: No comparisons between elements.

- ***Input***: $A[1 \,.\,.\, n]$, where $A[j] \in \{1, 2, \ldots, k\}$.
- ***Output***: $B[1 \,.\,.\, n]$, sorted.
- ***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 $C[A[j]] \leftarrow C[A[j]] + 1$      $\triangleright C[i] = |\{\text{key} = i\}|$
for $i \leftarrow 2$ to $k$
    do $C[i] \leftarrow C[i] + C[i-1]$       $\triangleright C[i] = |\{\text{key} \le i\}|$
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

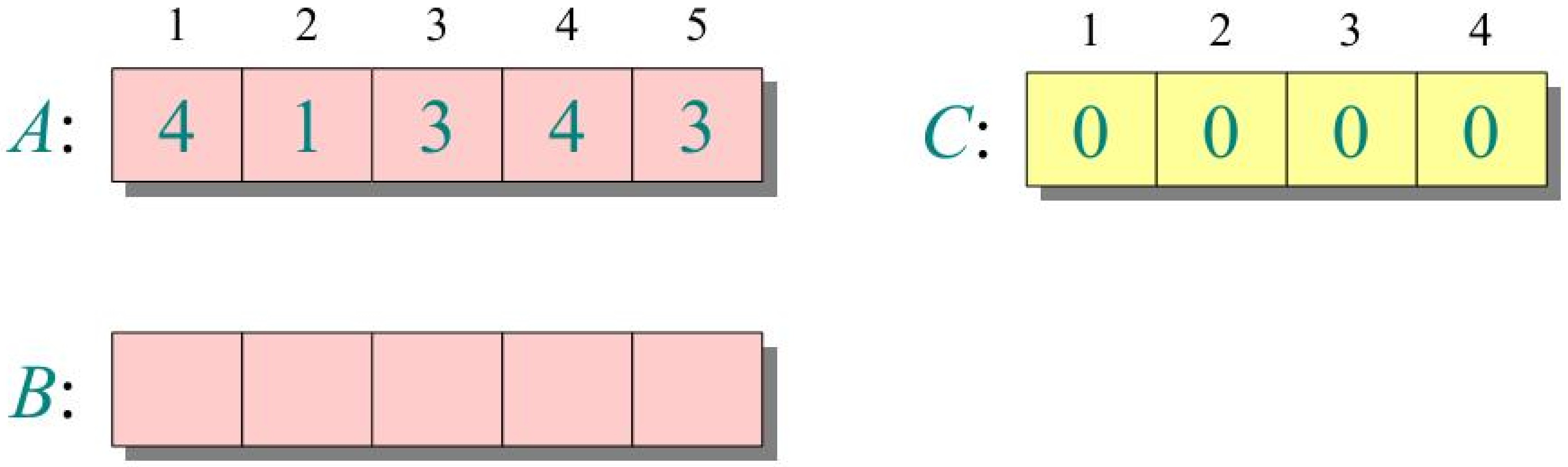
Counting-sort example

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

	1	2	3	4
C:				

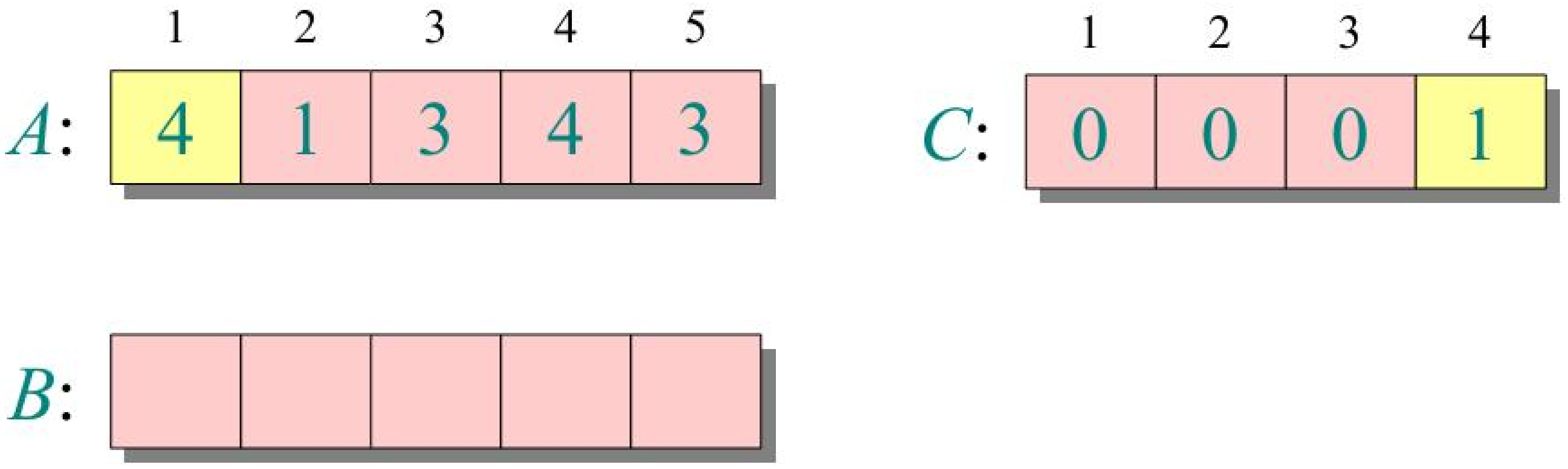
B:					

Loop 1



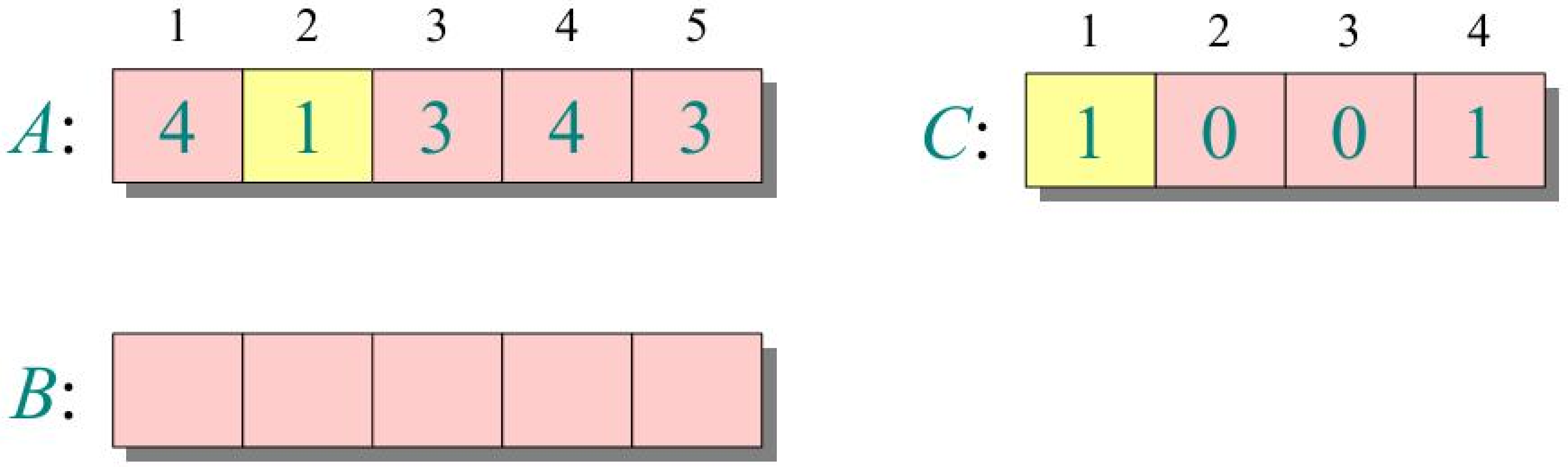
```
for $i \leftarrow 1$ to $k$
    do $C[i] \leftarrow 0$
```

Loop 2



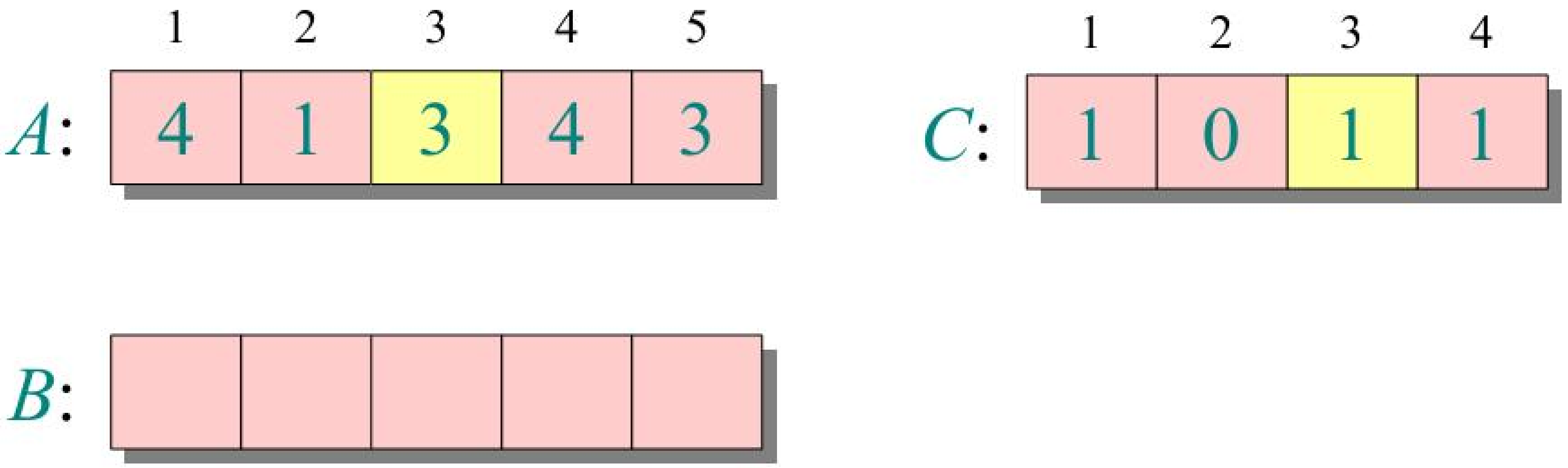
```
for $j \leftarrow 1$ to $n$
    do $C[A[j]] \leftarrow C[A[j]] + 1$    ▹ $C[i] = |\{\text{key} = i\}|$
```

Loop 2



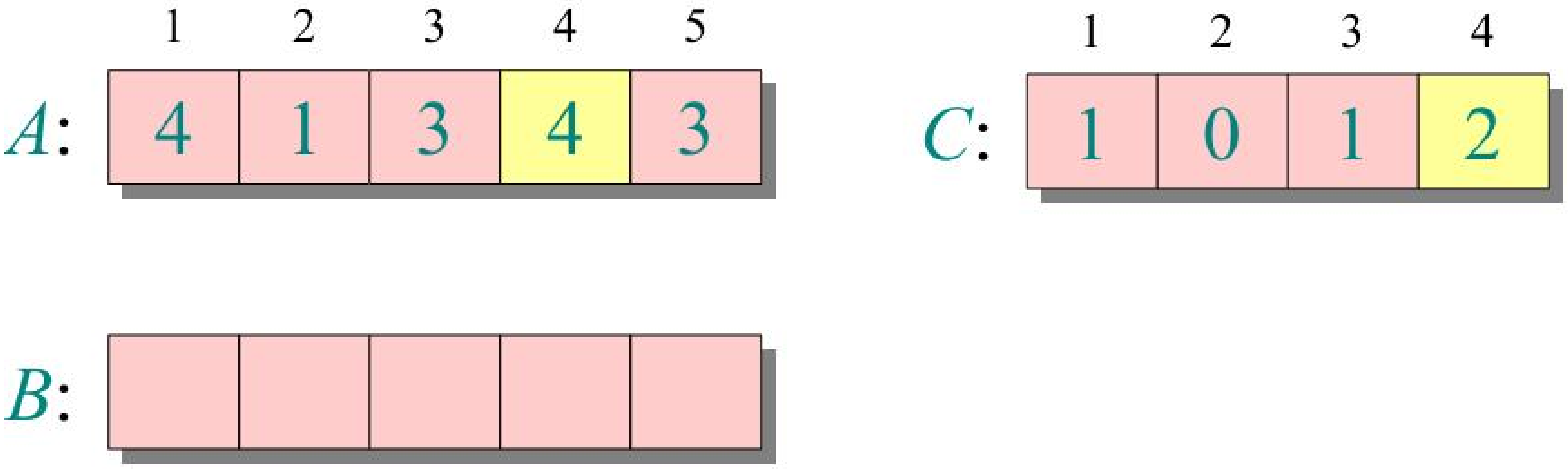
```
for $j \leftarrow 1$ to $n$
    do $C[A[j]] \leftarrow C[A[j]] + 1$    $\triangleright\ C[i] = |\{\text{key} = i\}|$
```

Loop 2



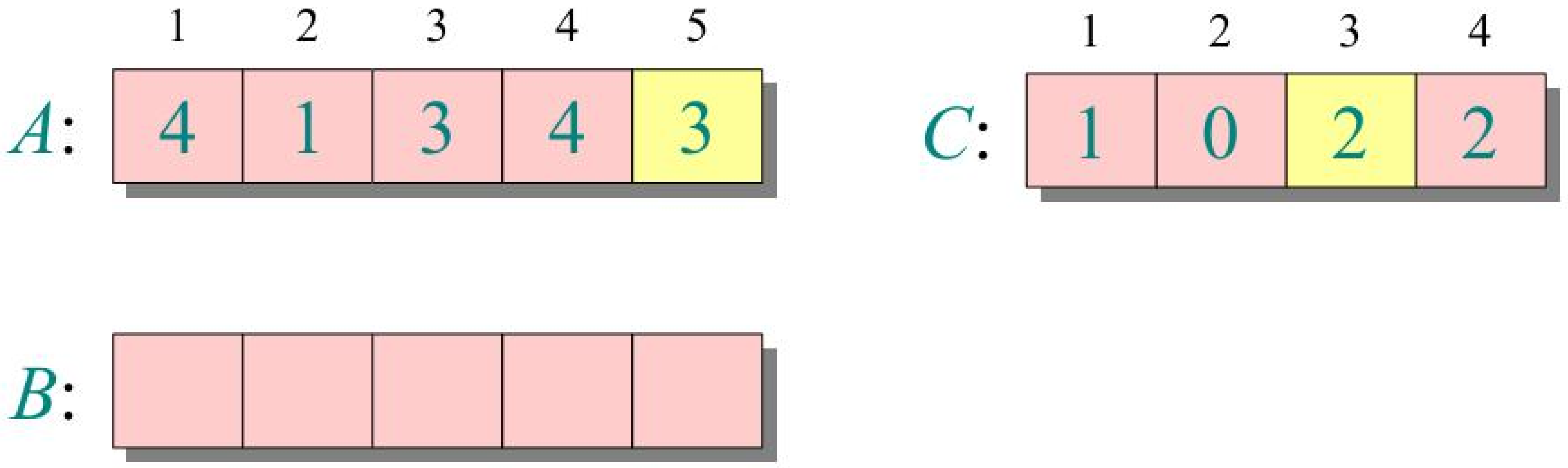
```
for $j \leftarrow 1$ to $n$
    do $C[A[j]] \leftarrow C[A[j]] + 1$    $\triangleright$ $C[i] = |\{\text{key} = i\}|$
```

Loop 2



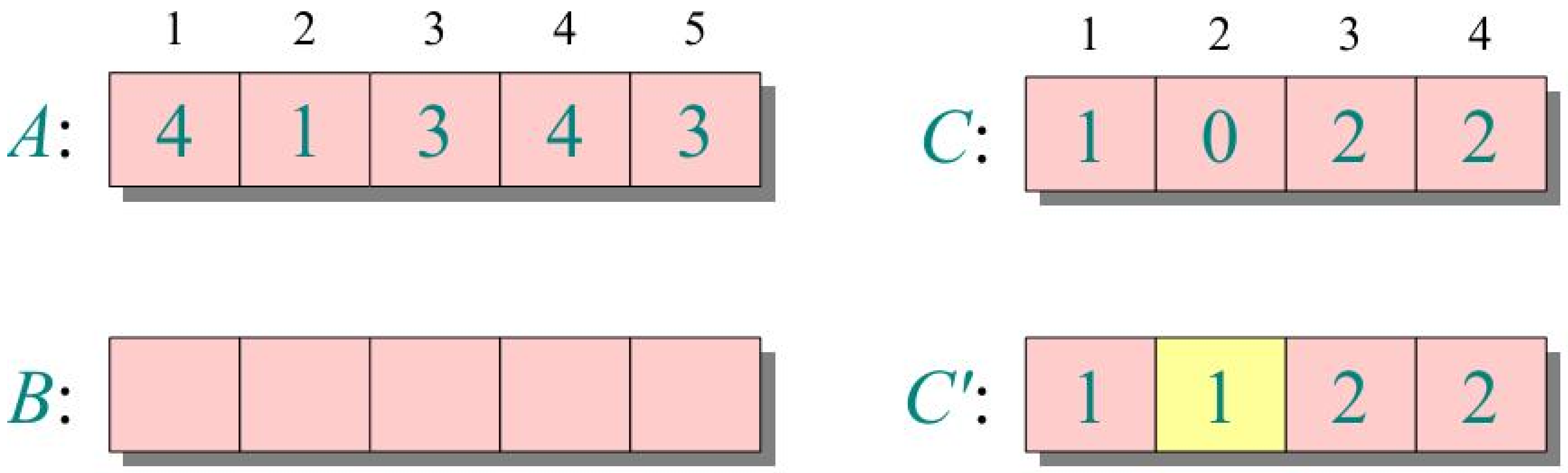
```
for $j \leftarrow 1$ to $n$
    do $C[A[j]] \leftarrow C[A[j]] + 1$    $\triangleright$ $C[i] = |\{\text{key} = i\}|$
```

Loop 2



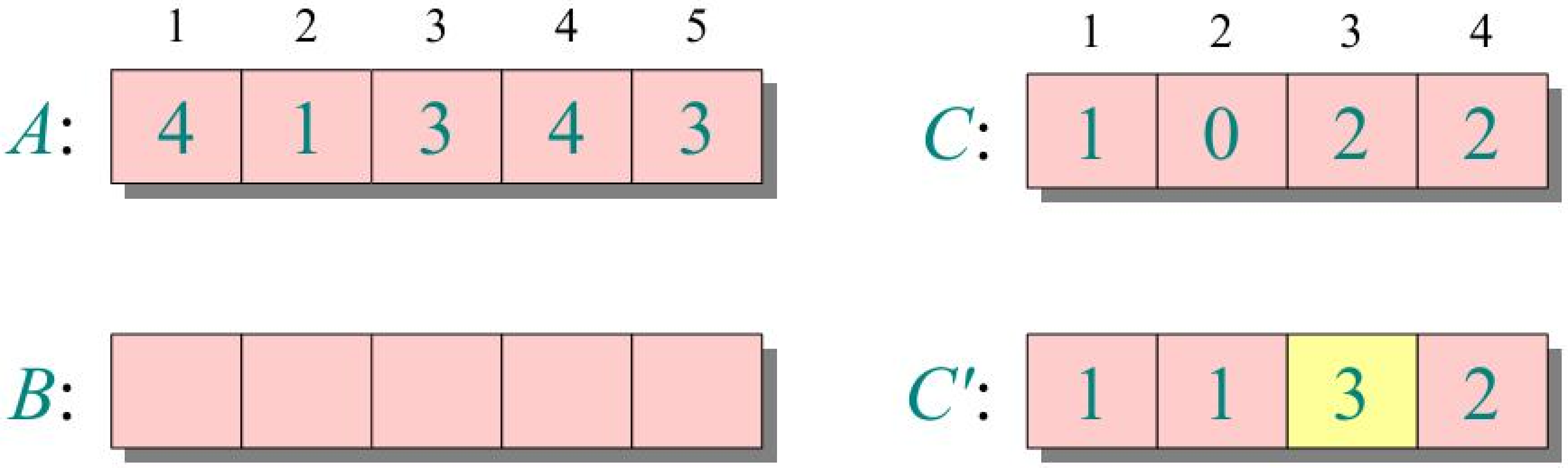
```
for $j \leftarrow 1$ to $n$
    do $C[A[j]] \leftarrow C[A[j]] + 1$    $\triangleright\ C[i] = |\{\text{key} = i\}|$
```

Loop 3



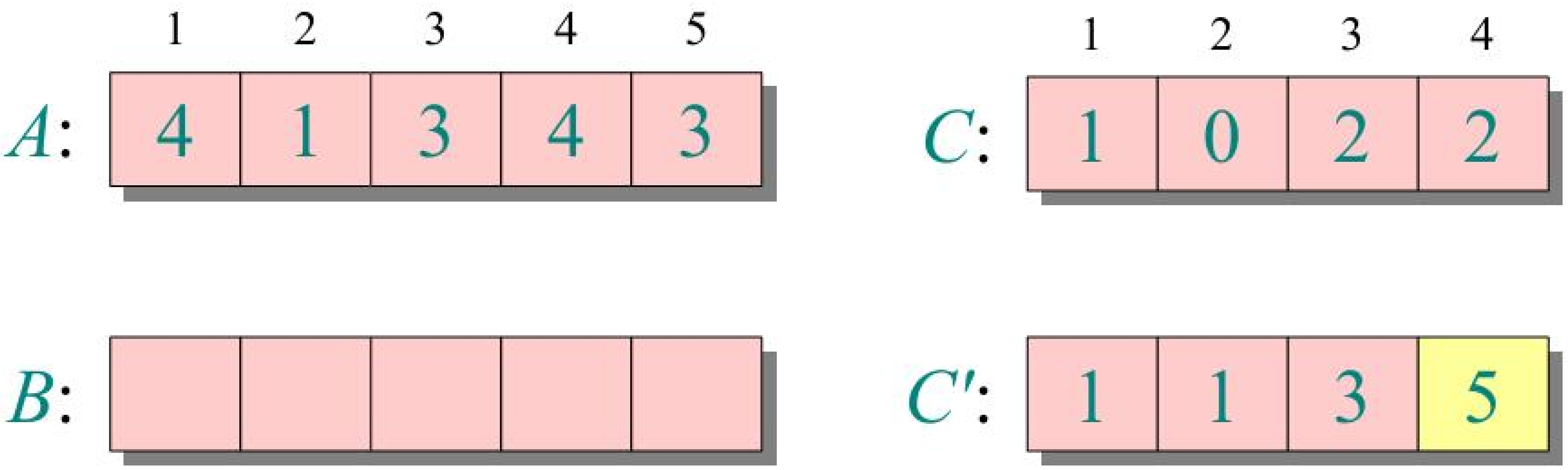
```
for $i \leftarrow 2$ to $k$
    do $C[i] \leftarrow C[i] + C[i-1]$        $\triangleright\ C[i] = |\{\text{key} \le i\}|$
```

Loop 3



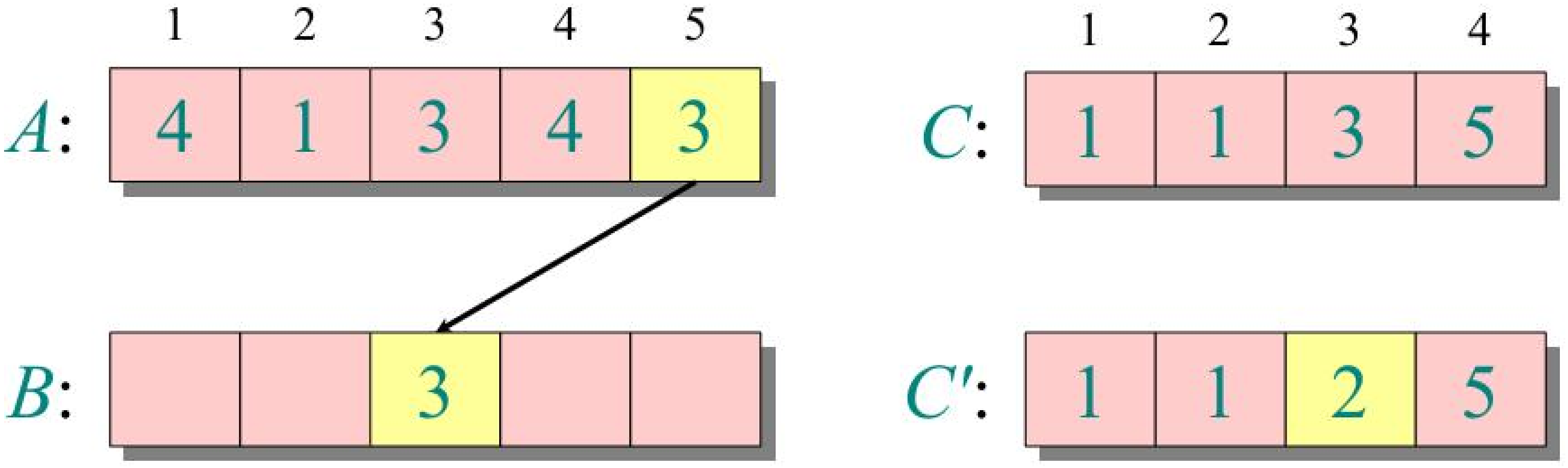
```
for $i \leftarrow 2$ to $k$
    do $C[i] \leftarrow C[i] + C[i-1]$        $\triangleright$ $C[i] = |\{\text{key} \le i\}|$
```

Loop 3



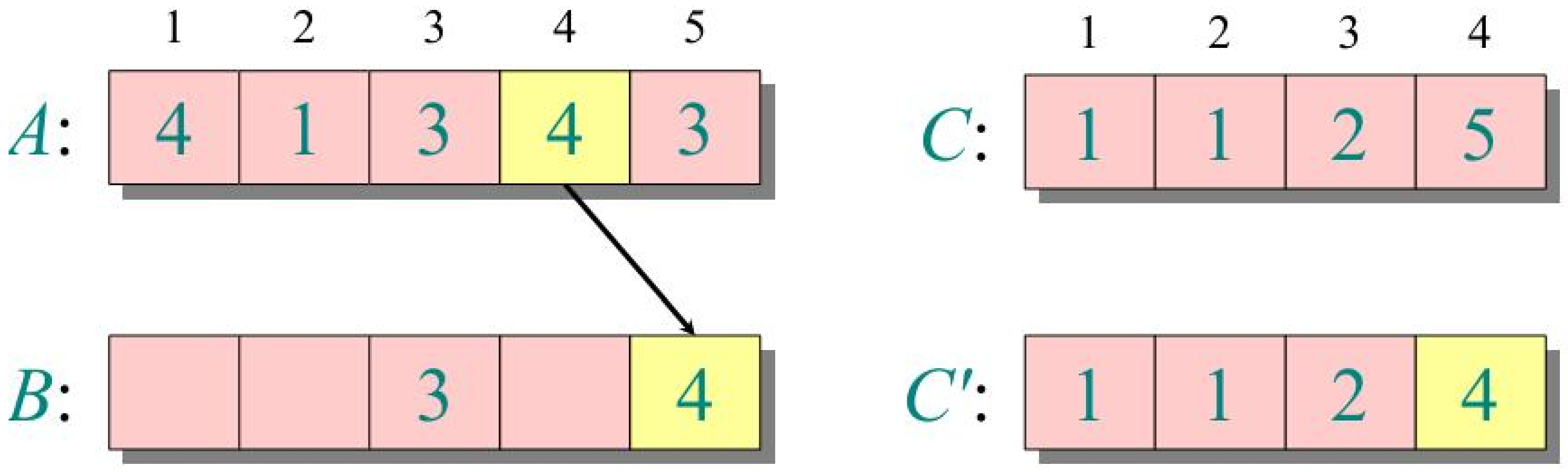
```
for $i \leftarrow 2$ to $k$
    do $C[i] \leftarrow C[i] + C[i-1]$        ▹ $C[i] = |\{\text{key} \le i\}|$
```

Loop 4



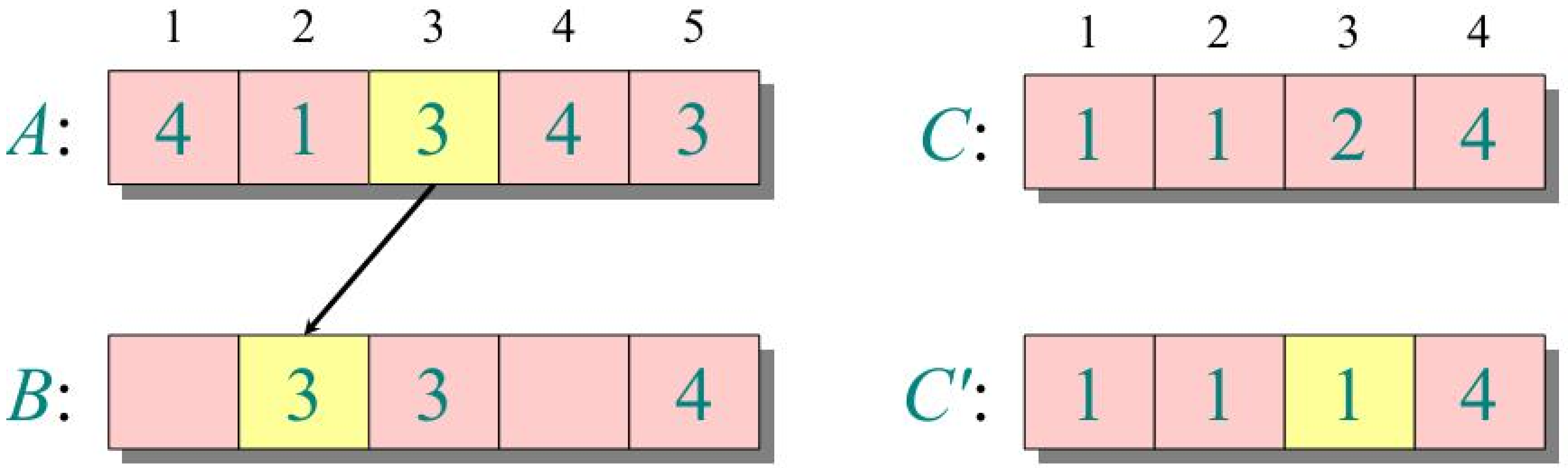
```
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow A[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4



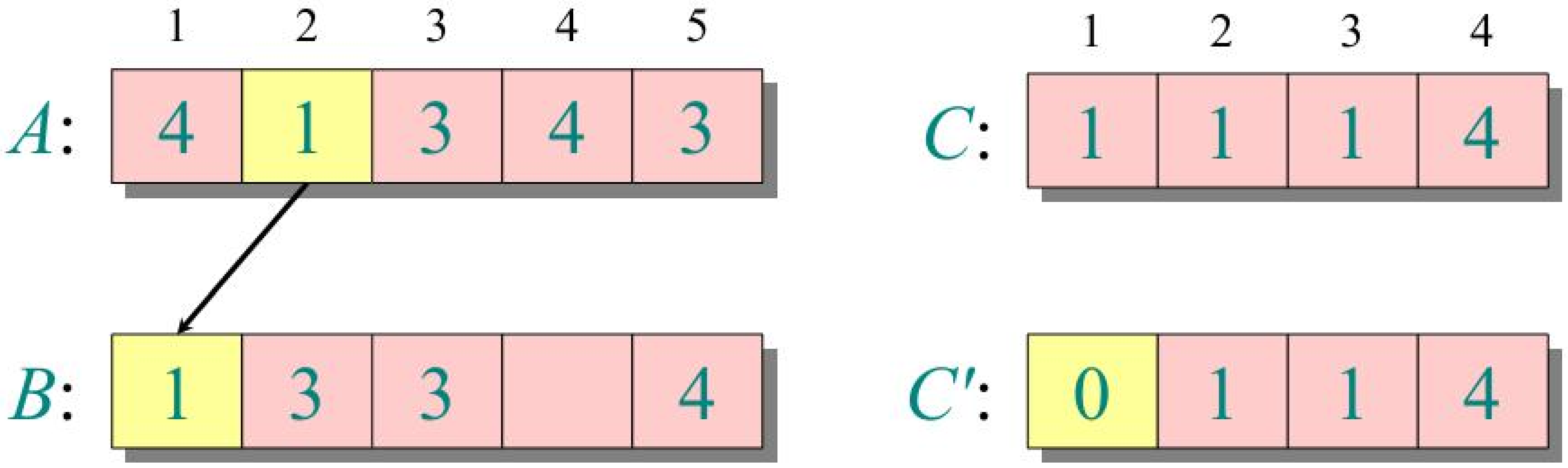
```
for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow A[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4



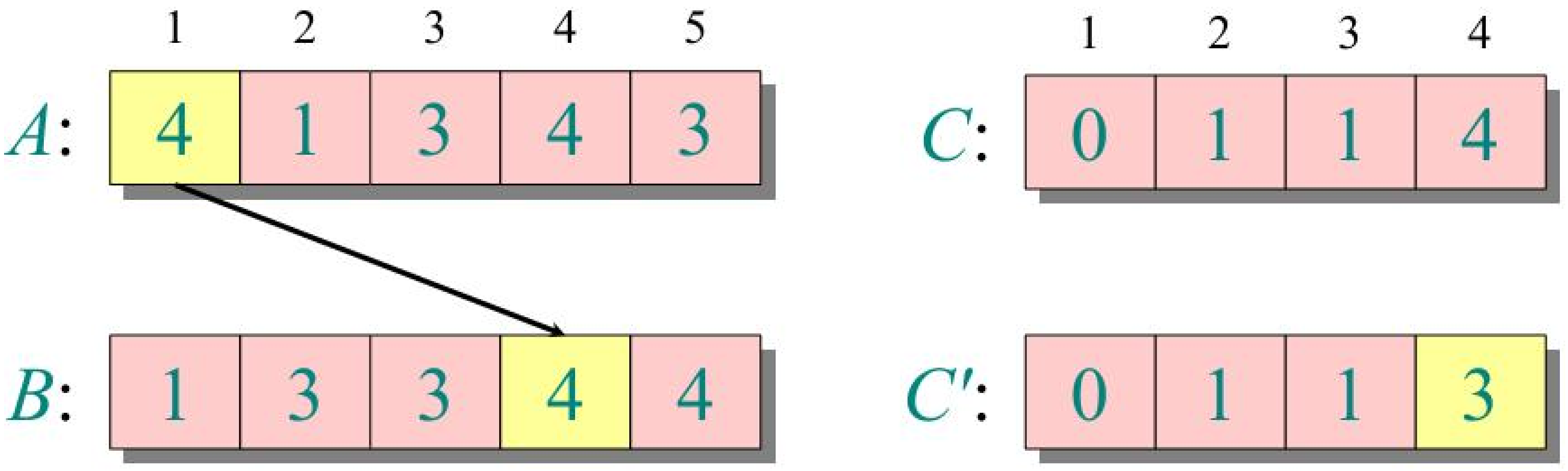
```
for $j \leftarrow n$ downto 1
    do $B[C[A[j]]] \leftarrow A[j]$
       $C[A[j]] \leftarrow C[A[j]] - 1$
```

Loop 4



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

Loop 4



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

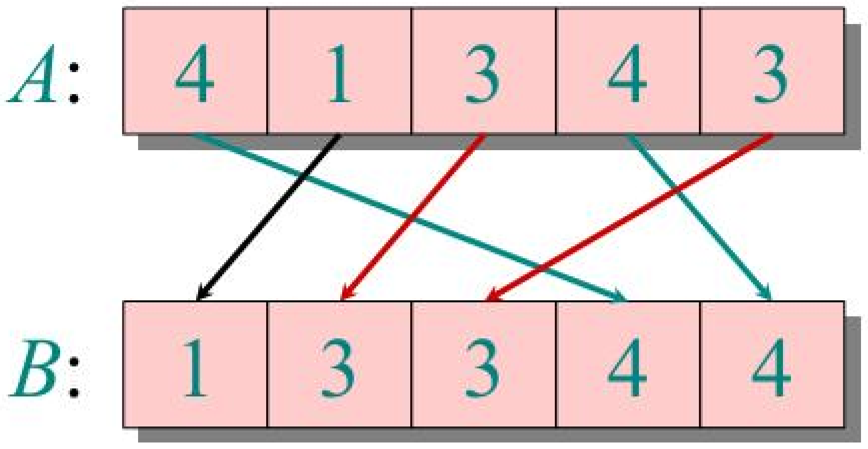
Analysis

```
$\Theta(k)$   { 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 $B[C[A[j]]] \leftarrow A[j]$
                       $C[A[j]] \leftarrow C[A[j]] - 1$
```

$\Theta(n + k)$

Stable sorting

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



作业：7.1-4，8.2-1

Thank You!

Q&A