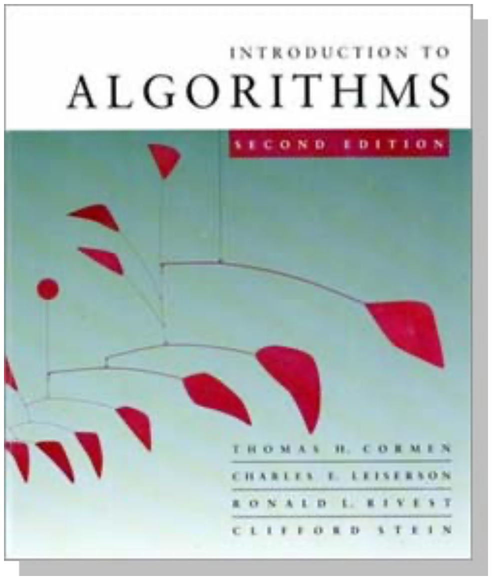
Introduction to Algorithms

6.046J/18.401J



LECTURE 5

Sorting Lower Bounds

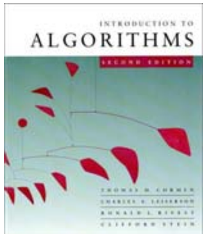
- Decision trees

Linear-Time Sorting

- Counting sort
- Radix sort

Appendix: Punched cards

Prof. Erik Demaine



How fast can we sort?

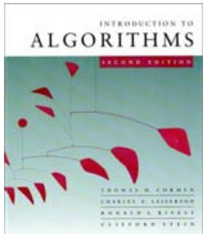
All the sorting algorithms we have seen so far are ***comparison sorts***: only use comparisons to determine the relative order of elements.

- *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.

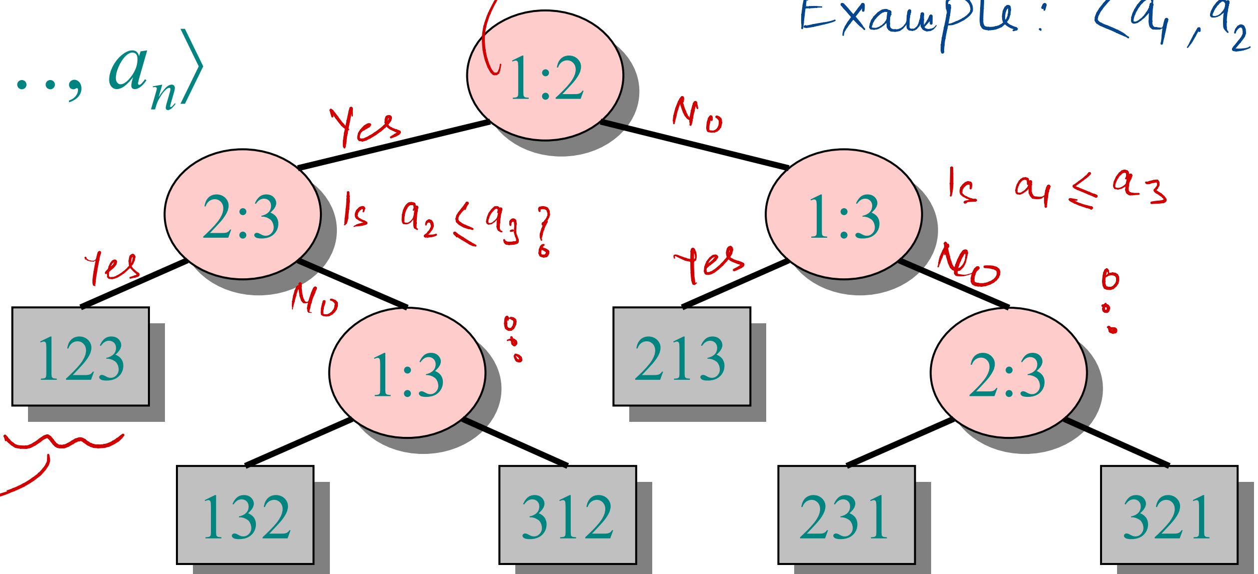


Decision-tree example

Is $a_1 \le a_2$?

Example: $\langle a_1, a_2, a_3 \rangle$

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



Is $a_2 \le a_3$?

Is $a_1 \le a_3$

$a_1 \le a_2$ & $a_2 < a_3$

$\Rightarrow a_1 \le a_2 \le a_3$

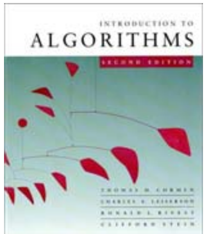
$\Rightarrow \langle 1, 2, 3 \rangle$

$a_1 \le a_2,\ a_2 > a_3,\ a_1 \le a_3$

$\Rightarrow a_1 \le a_3 \le a_2 \Rightarrow \langle 1, 3, 2 \rangle$

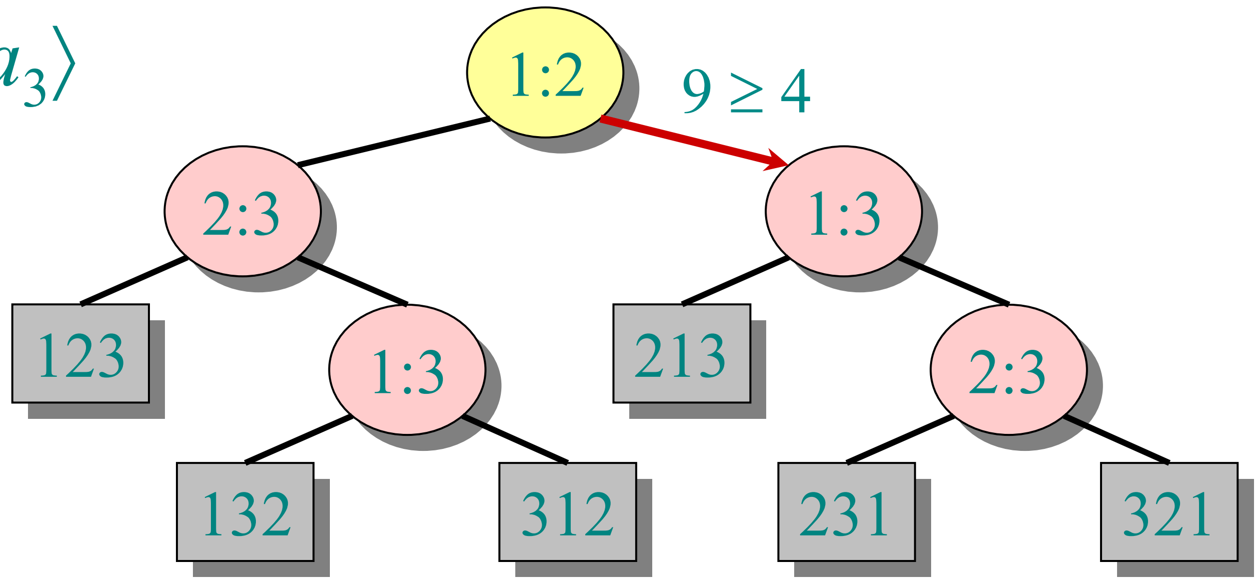
Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

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



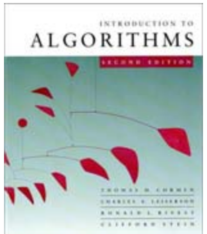
Decision-tree example

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



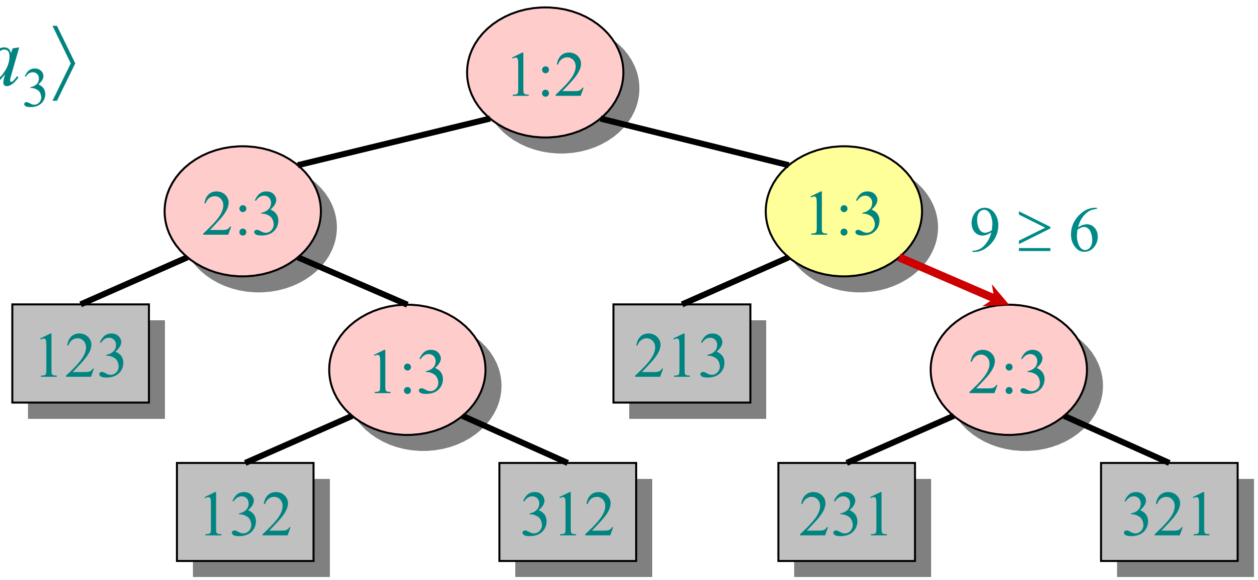
Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

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



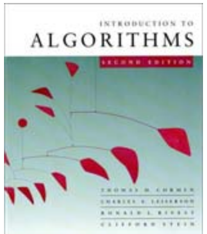
Decision-tree example

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



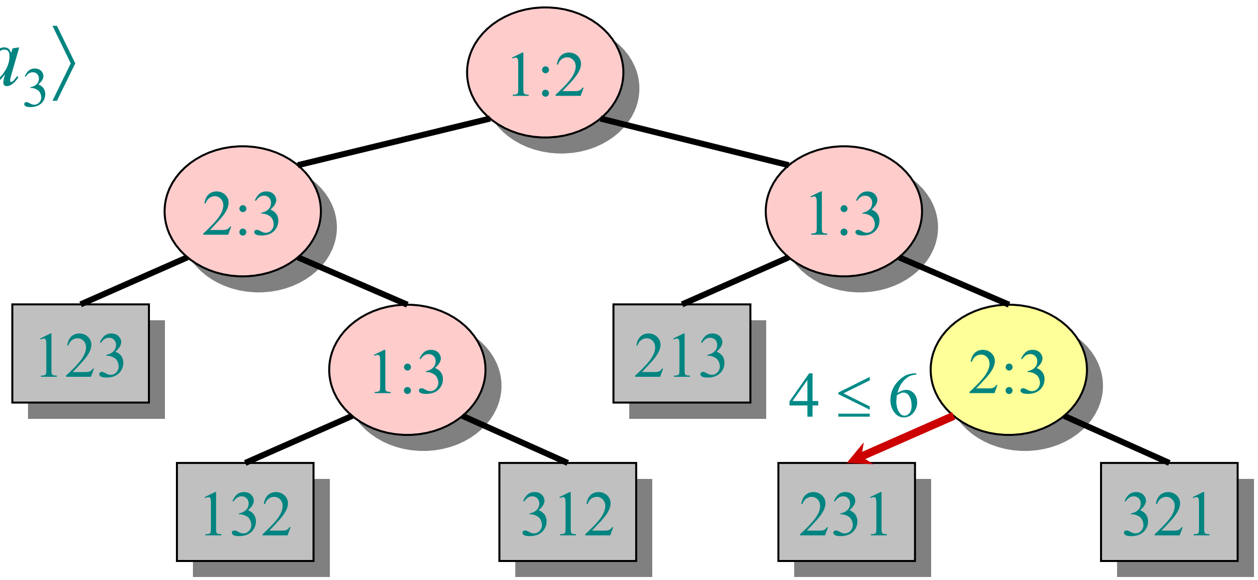
Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

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



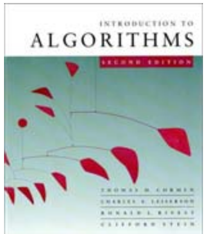
Decision-tree example

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



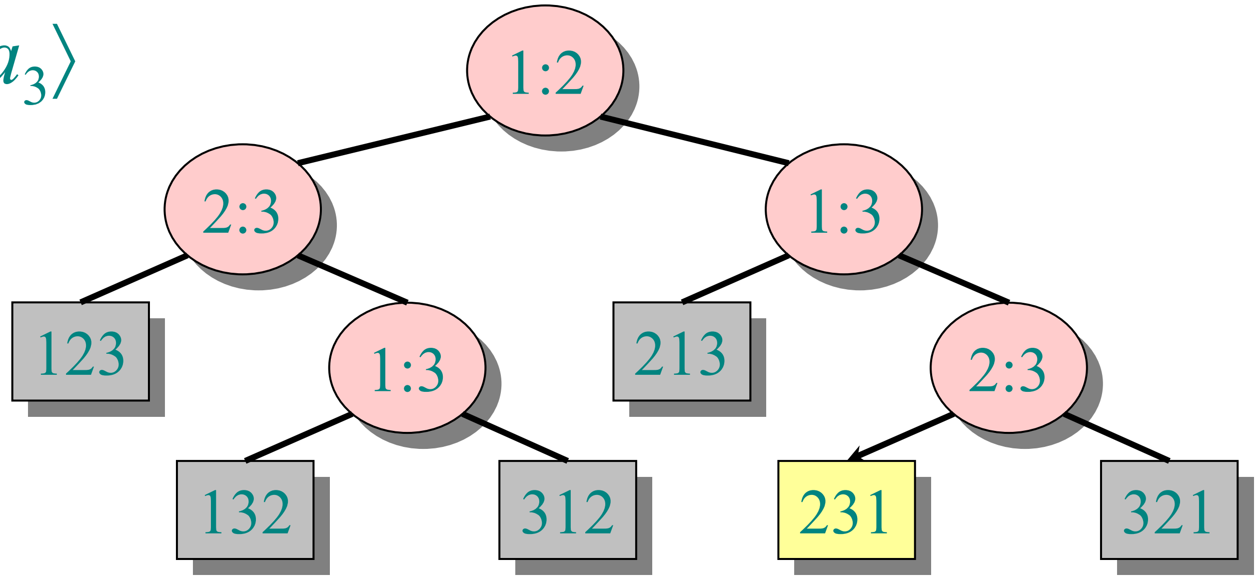
Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

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



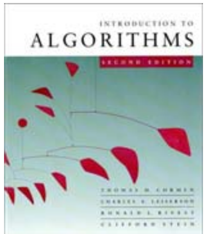
Decision-tree example

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



$4 \leq 6 \leq 9$

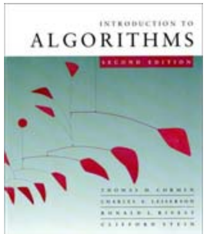
Each leaf contains a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ to indicate that the ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$ has been established.



Decision-tree model

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

- One tree for each input size n.
- 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 $\Omega(n \lg n)$.

$\boxed{l \geq n!}$ (1)

If the number of leaves in l, then

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, $n! \leq 2^h$.

$\boxed{l \leq 2^h}$ (2) $\Rightarrow$ $n! \leq l \leq 2^h$

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

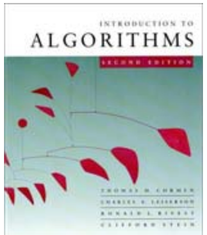
$\Rightarrow$ All comparison-based sorting algorithm cannot do better than $O(n \log n)$ in the worst case.

Sterling's Formula for approximating $\underline{n!}$

(a weak upperbound : $n! \leq n^n$)

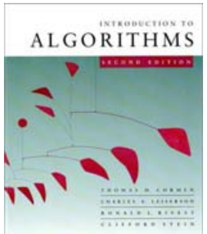
$$n! = \underbrace{\sqrt{2\pi n}}_{>1} \left(\frac{n}{e}\right)^n \underbrace{\left(1 + \theta\left(\frac{1}{n}\right)\right)}_{>1} \quad ; \; n \text{ positive}$$

$$\Rightarrow \quad n! \geq \left(\frac{n}{e}\right)^n$$



Lower bound for comparison sorting

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

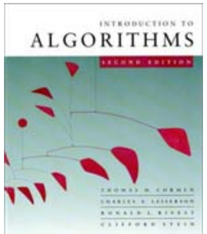


Sorting in linear time

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***: $C[1 \,.\,.\, k]$.

constraint !



Counting sort

Example of Prefix sum

a: | 1 | 7 | 6 | 3 | 0 |

P: | 1 | 8 | 14 | 17 | 17 |

Prefix sum array

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

  do $C[i] \leftarrow 0$

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

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

Increment C[A[j]]

a comment

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

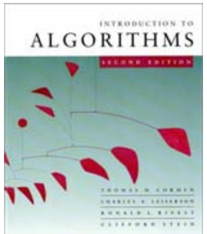
  do $C[i] \leftarrow C[i] + C[i-1]$ $\triangleright$ $C[i] = |\{\text{key} \le i\}|$

Prefix sum

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

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

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

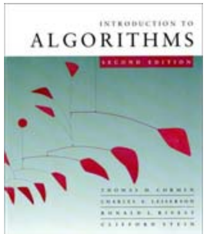


Counting-sort example

$k = 4$

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

	1	2	3	4
C:				



Loop 1

A:

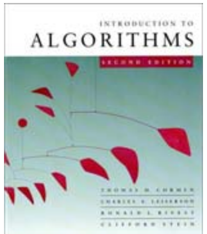
1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
0	0	0	0

B:

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



Loop 2

A:

1	2	3	4	5
4	1	3	4	3

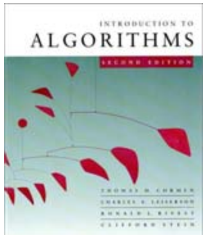
C:

1	2	3	4
0	0	0	1

B:

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

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



Loop 2

A:

1	2	3	4	5
4	1	3	4	3

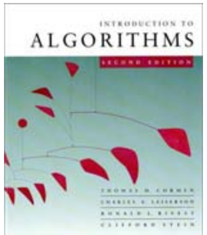
C:

1	2	3	4
1	0	0	1

B:

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

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



Loop 2

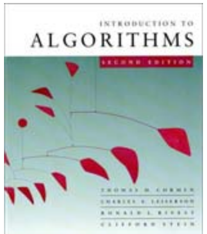
	1	2	3	4	5
A:	4	1	3	4	3

	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

A:

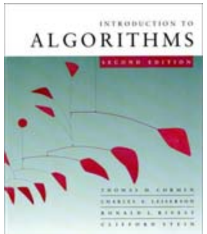
1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
1	0	1	2

B:

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



Loop 2

A:

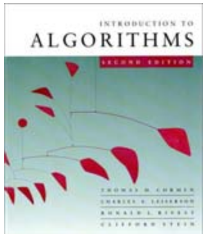
1	2	3	4	5
4	1	3	4	3

C:

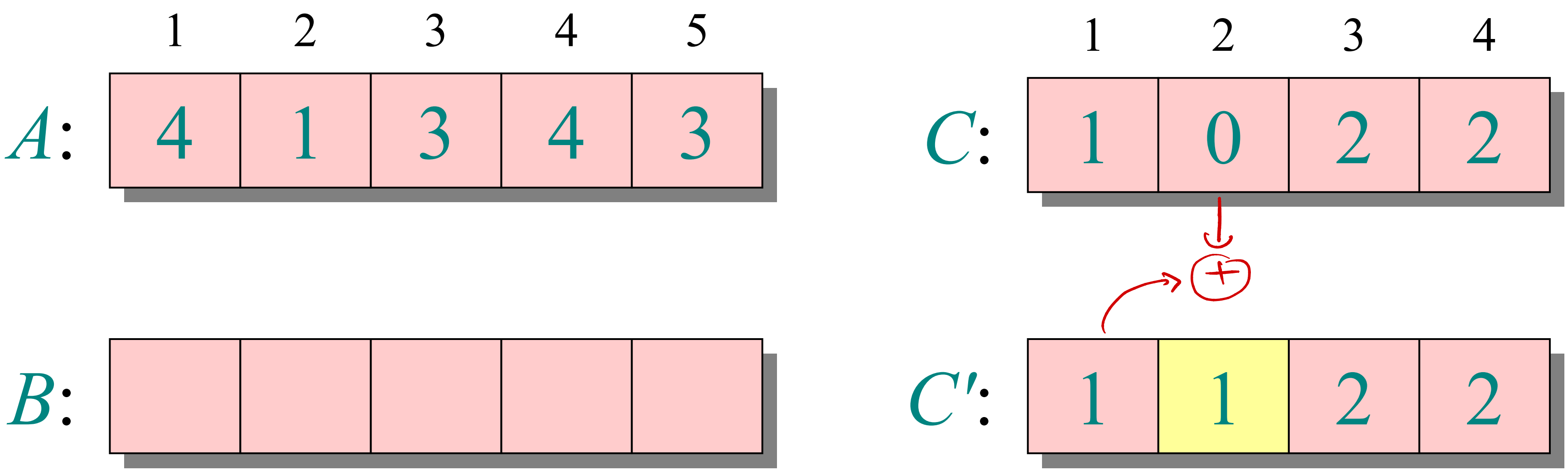
1	2	3	4
1	0	2	2

B:

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

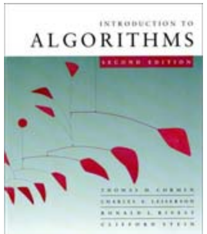


Loop 3



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

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



Loop 3

A:

1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
1	0	2	2

+

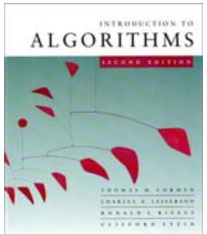
B:

C′:

1	1	3	2

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

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



Loop 3

A:

1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
1	0	2	2

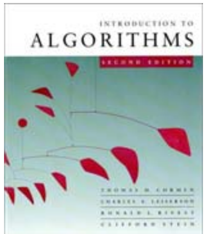
B:

C':

1	1	3	5

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

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



Loop 4

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

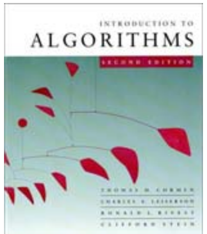
Prefix sum array

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

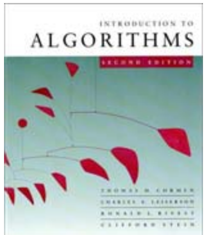
	1	2	3	4	5
A:	4	1	3	4	3
B:			3		4

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

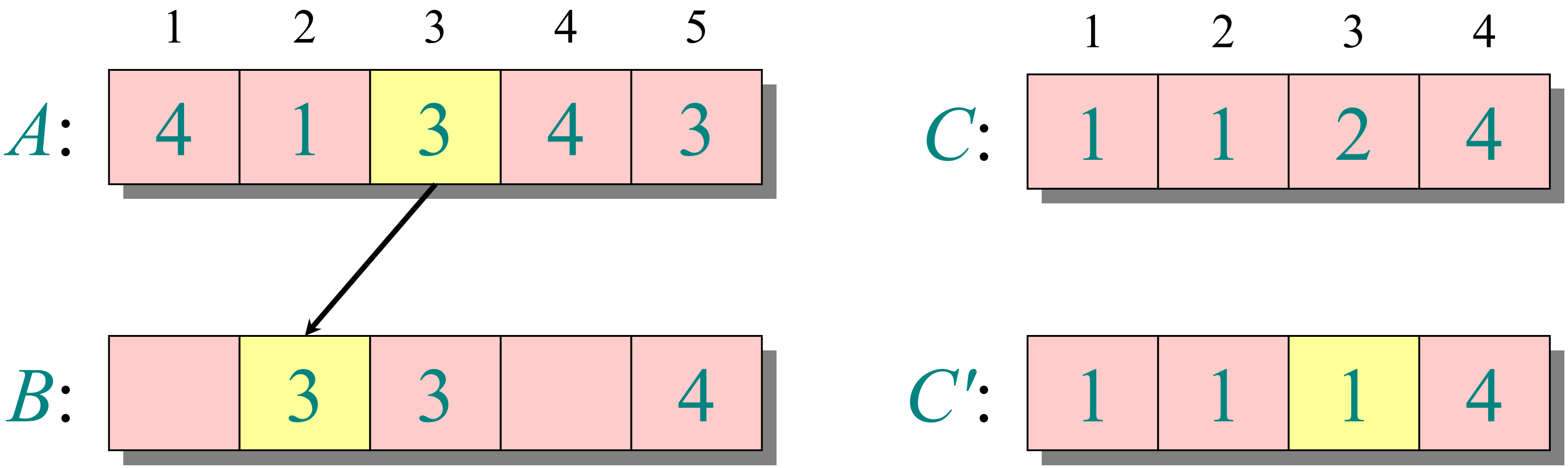
for $j \leftarrow n$ **downto** 1

do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$

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



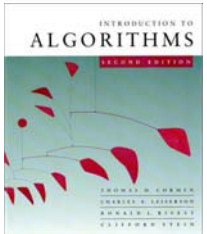
Loop 4



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

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

	1	2	3	4
C:	1	1	1	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

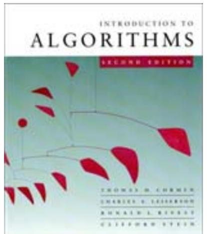
	1	2	3	4	5
A:	4	1	3	4	3
B:	1	3	3	4	4

	1	2	3	4
C:	0	1	1	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

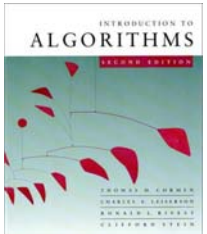
$\Theta(k)$
for $i \leftarrow 1$ **to** k
  do $C[i] \leftarrow 0$

$\Theta(n)$
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 $B[C[A[j]]] \leftarrow A[j]$
    $C[A[j]] \leftarrow C[A[j]] - 1$

$\Theta(n + k)$



Running time

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

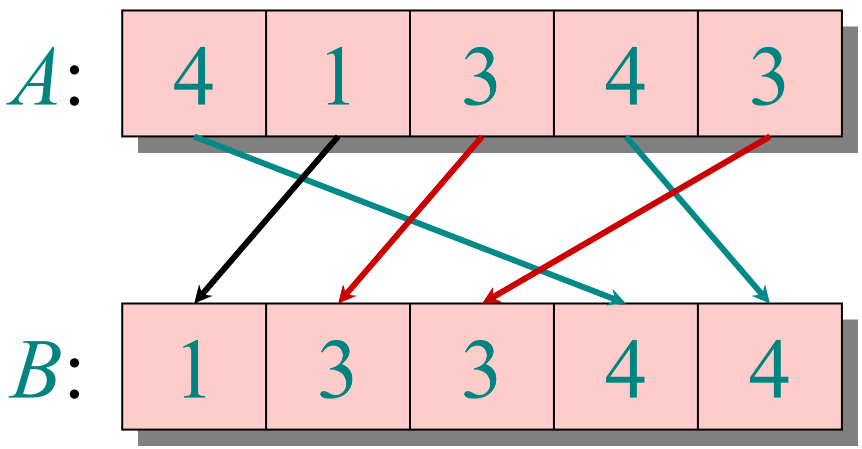
- But, sorting takes $\Omega(n \lg n)$ time!
- Where's the fallacy?

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.



Exercise: What other sorts have this property?

