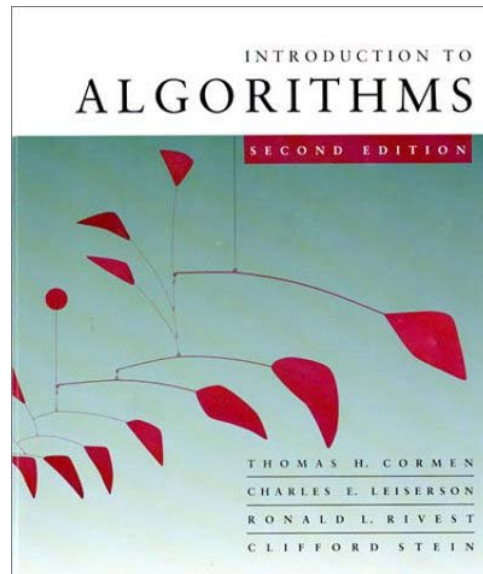


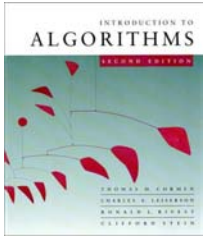
Introduction to Algorithms

6.046J/18.401J



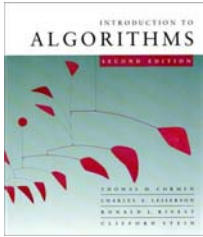
Lecture 6

Prof. Piotr Indyk



Today: sorting

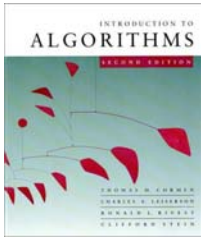
- Show that $\Theta(n \lg n)$ is the best possible running time for a sorting algorithm.
- Design an algorithm that sorts in $O(n)$ time.
- Hint: different models ?



Comparison sort

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

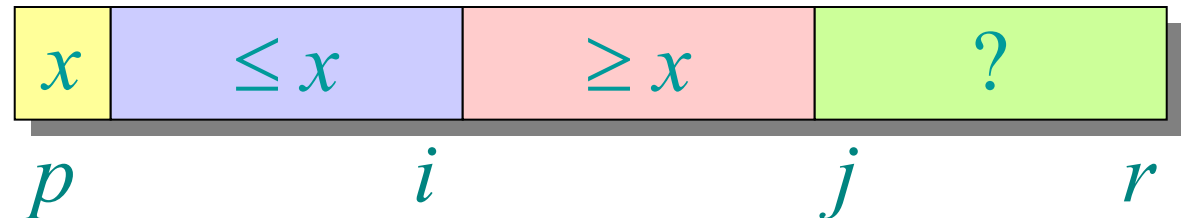
- *E.g.*, insertion sort, merge sort, quicksort, heapsort.



Partitioning subroutine

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

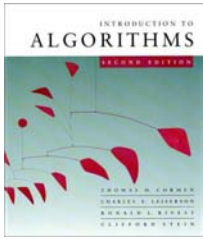
Invariant:





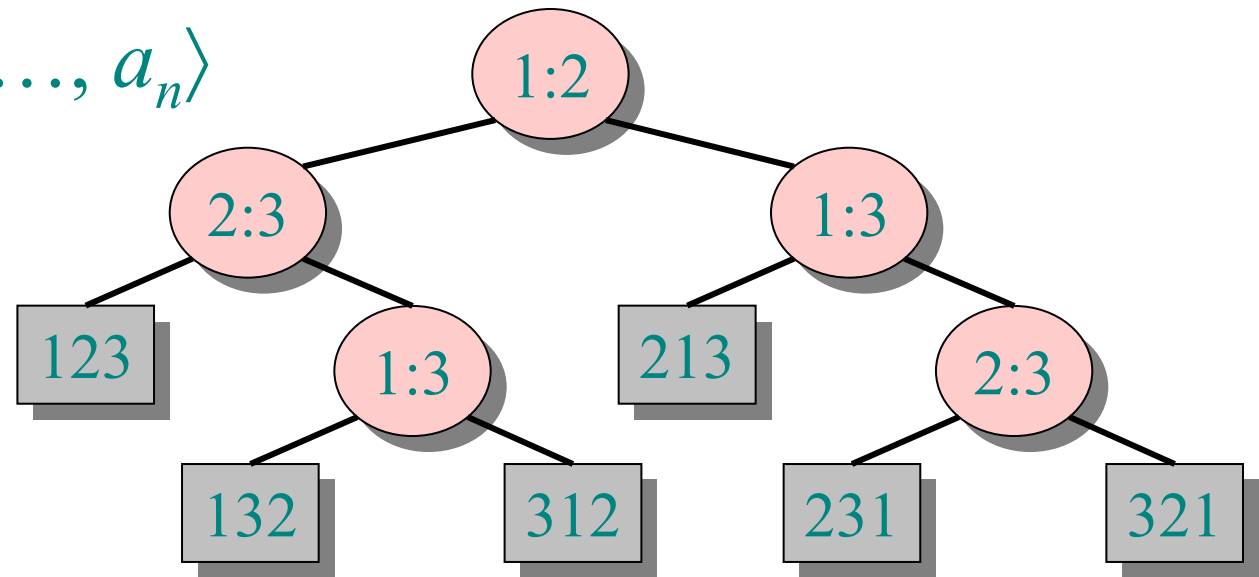
Comparison sort

- All of our algorithms used comparisons
- All of our algorithms have running time $\Omega(n \lg n)$
- Is it the best that we can do using just comparisons ?
- Answer: YES, via *decision trees*



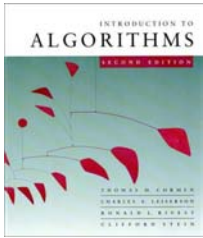
Decision-tree example

Sort $\langle a_1, a_2, \dots, a_n \rangle$
($n=3$)



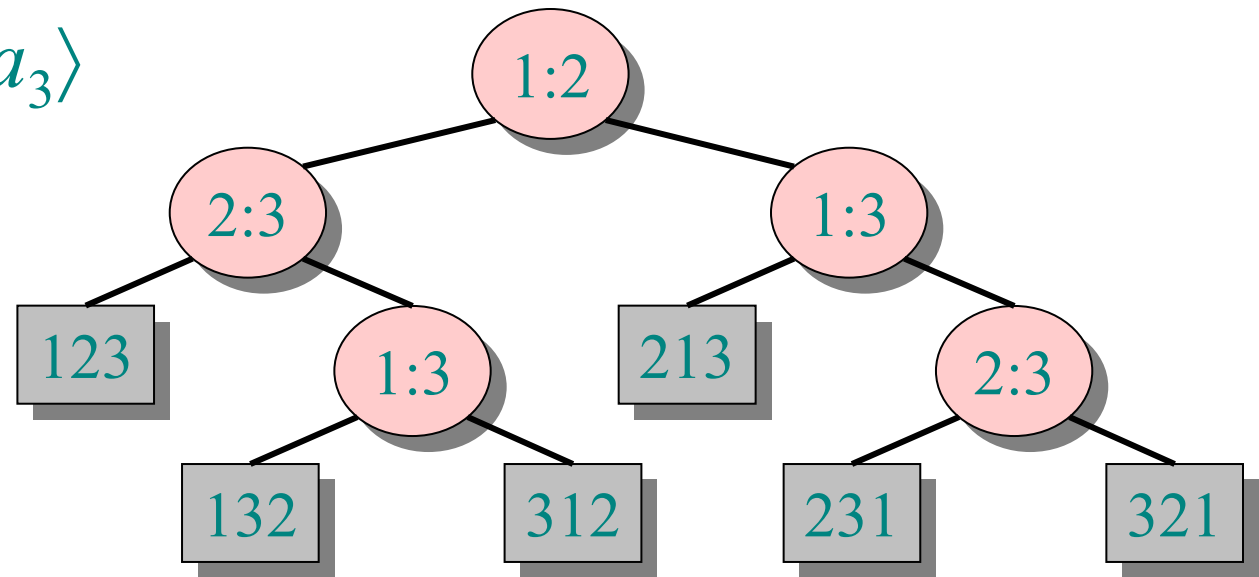
Each internal node is labeled $i:j$ for $i, j \in \{1, 2, \dots, n\}$.

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



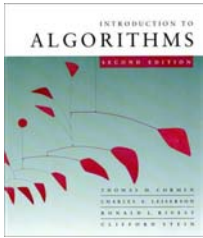
Decision-tree example

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



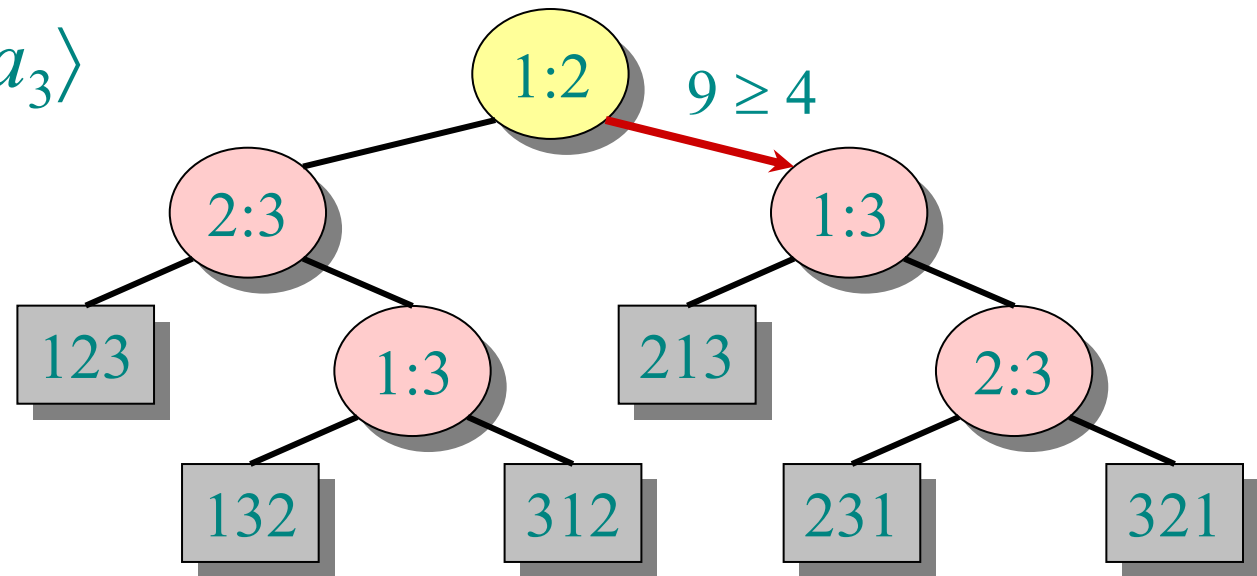
Each internal node is labeled $i:j$ for $i, j \in \{1, 2, \dots, n\}$.

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



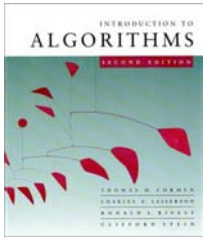
Decision-tree example

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



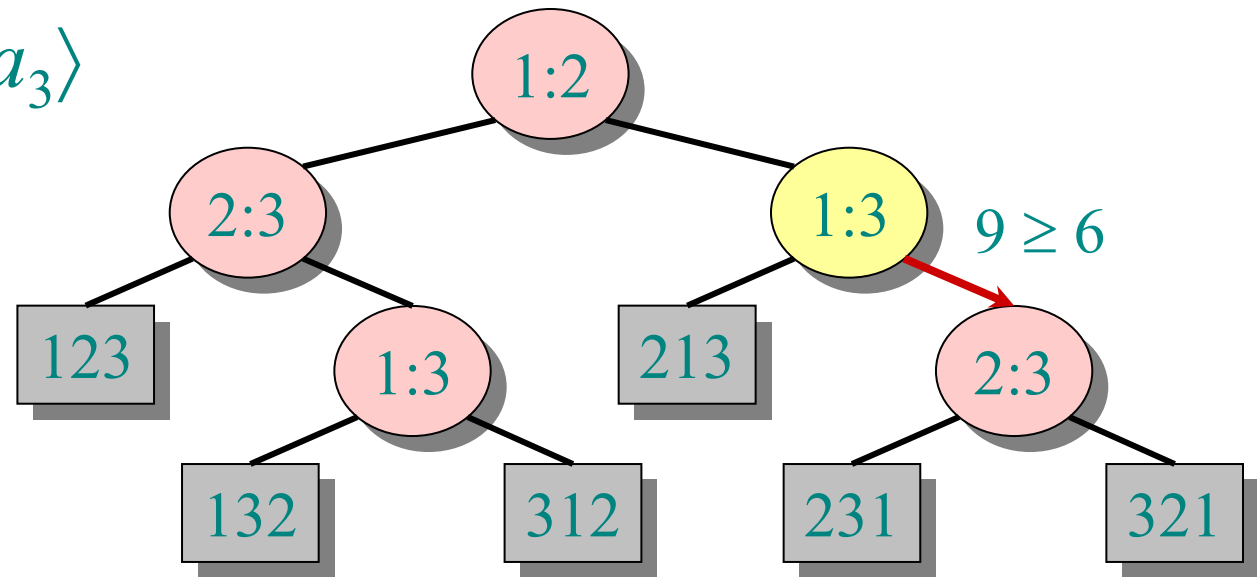
Each internal node is labeled $i:j$ for $i, j \in \{1, 2, \dots, n\}$.

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



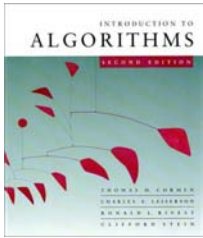
Decision-tree example

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



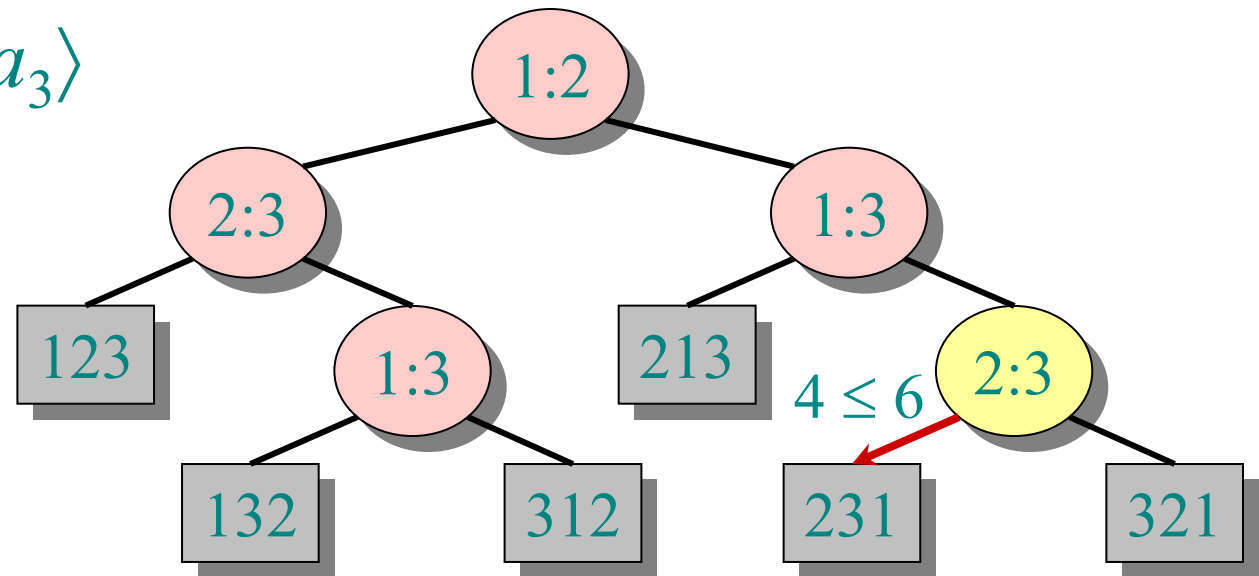
Each internal node is labeled $i:j$ for $i, j \in \{1, 2, \dots, n\}$.

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



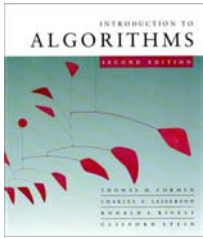
Decision-tree example

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



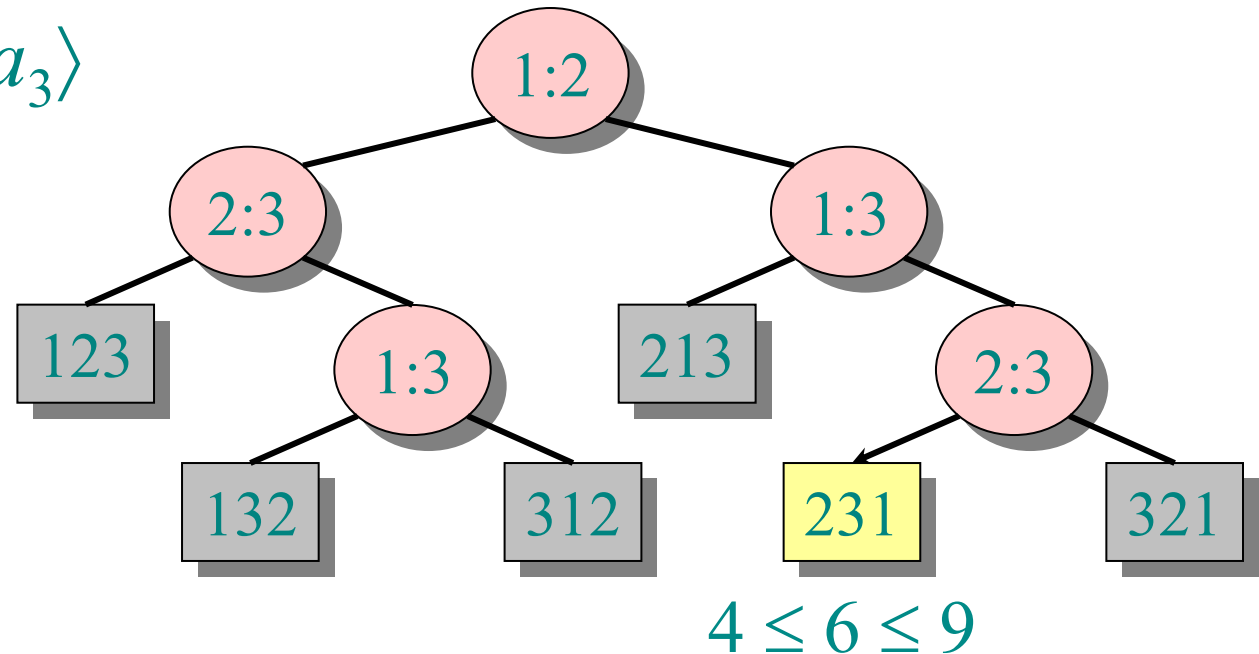
Each internal node is labeled $i:j$ for $i, j \in \{1, 2, \dots, n\}$.

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

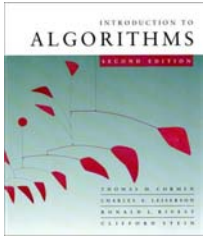


Decision-tree example

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



Each leaf contains a permutation $\langle \pi(1), \pi(2), \dots, \pi(n) \rangle$ to indicate that the ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$ 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 number of comparisons done by the algorithm on a given input =
...the length of the path taken.
- Worst-case number of comparisons =
...max path length = height of tree.
- Worst-case time \geq worst-case number of comparisons

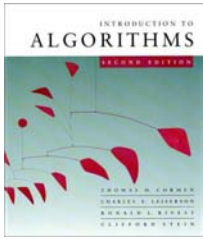


Lower bound for decision-tree sorting

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

Corollary. Any comparison sorting algorithm has worst-case running time $\Omega(n \lg n)$.

Corollary 2. Merge sort and Heap Sort are asymptotically optimal comparison sorting algorithms.

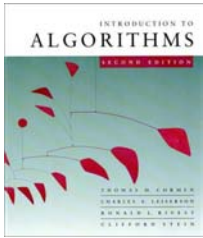


Lower bound for decision-tree sorting

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, $2^h \geq \text{\#leaves} \geq n!$, or $h \geq \lg(n!)$



Proof, ctd.

$$\begin{aligned} 2^h &\geq n! \\ &\geq n * (n-1) * \dots * \lceil n/2 \rceil \\ &\geq (n/2)^{n/2} \\ \Rightarrow h &\geq \lg((n/2)^{n/2}) \\ &\geq (n/2) (\lg n - \lg 2) \\ &= \Omega(n \lg n). \end{aligned}$$

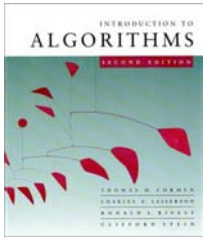




Example: sorting 3 elements

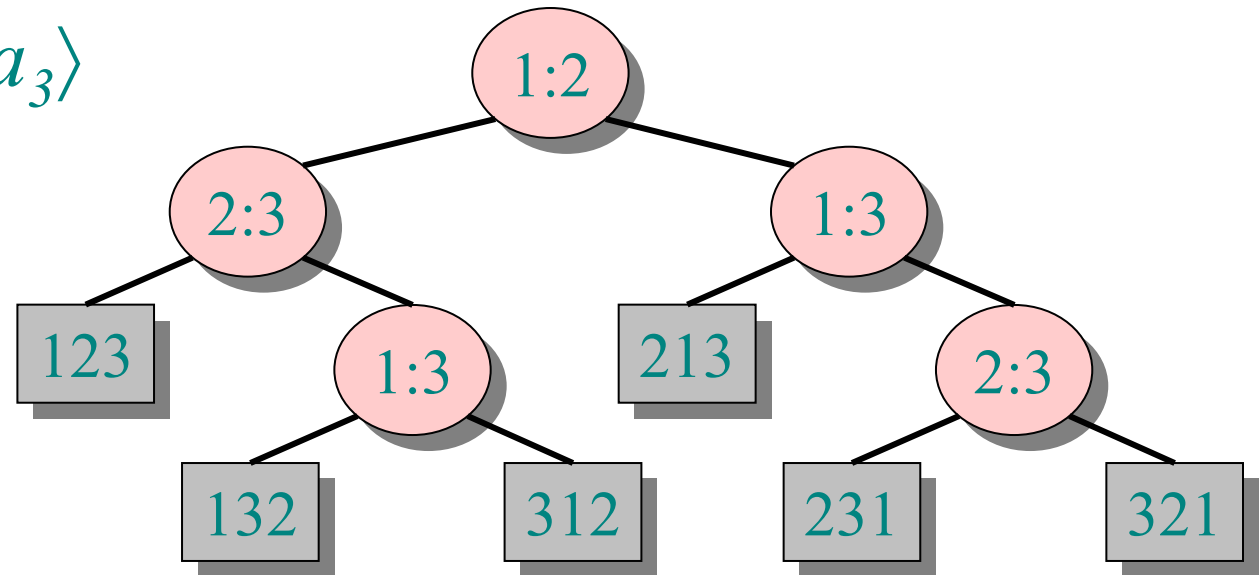
Recall $h \geq \lg(n!)$

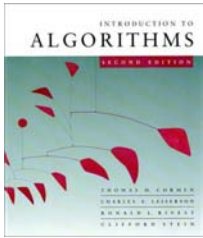
- $n=3$
- $n!=6$
- $\log_2 6 = 2.58$
- Sorting 3 elements requires...
... ≥ 3 comparisons in the worst case



Decision-tree for $n=3$

Sort $\langle a_1, a_2, a_3 \rangle$



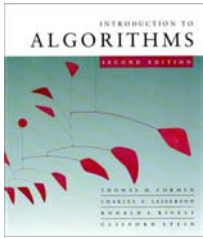


Sorting in linear time

Counting sort: No comparisons between elements.

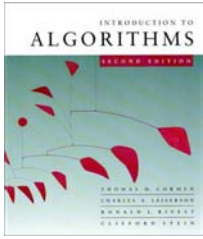
- **Input:** $A[1 \dots n]$, where $A[j] \in \{1, 2, \dots, k\}$.
- **Output:** $B[1 \dots n]$, sorted*
- **Auxiliary storage:** $C[1 \dots k]$.

*Actually, we require the algorithm to construct a **permutation** of the input array A that produces the sorted array B. This permutation can be obtained by making small changes to the last loop of the algorithm.

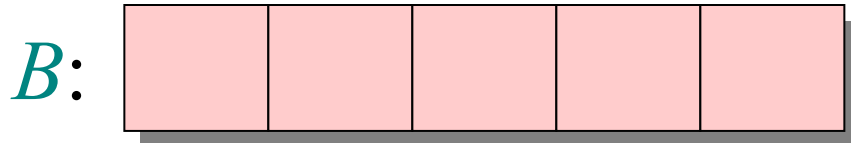
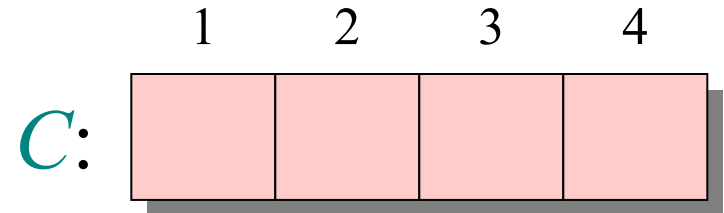
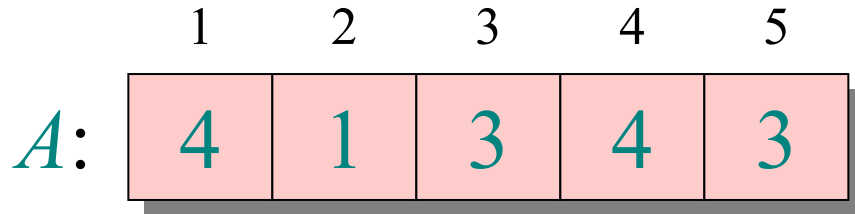


Counting sort

```
for  $i \leftarrow 1$  to  $k$ 
  do  $C[i] \leftarrow 0$ 
for  $j \leftarrow 1$  to  $n$ 
  do  $C[A[j]] \leftarrow C[A[j]] + 1$      $\triangleleft C[i] = |\{\text{key} = i\}|$ 
for  $i \leftarrow 2$  to  $k$ 
  do  $C[i] \leftarrow C[i] + C[i-1]$      $\triangleleft C[i] = |\{\text{key} \leq i\}|$ 
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





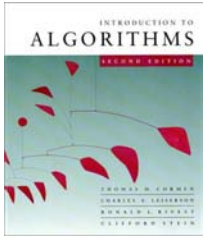
Loop 1

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

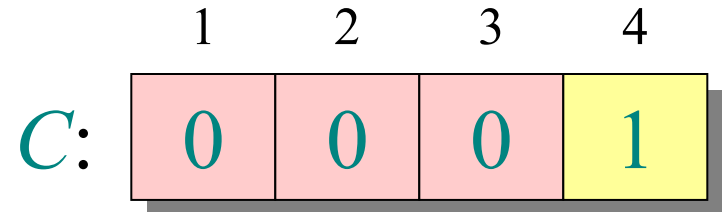
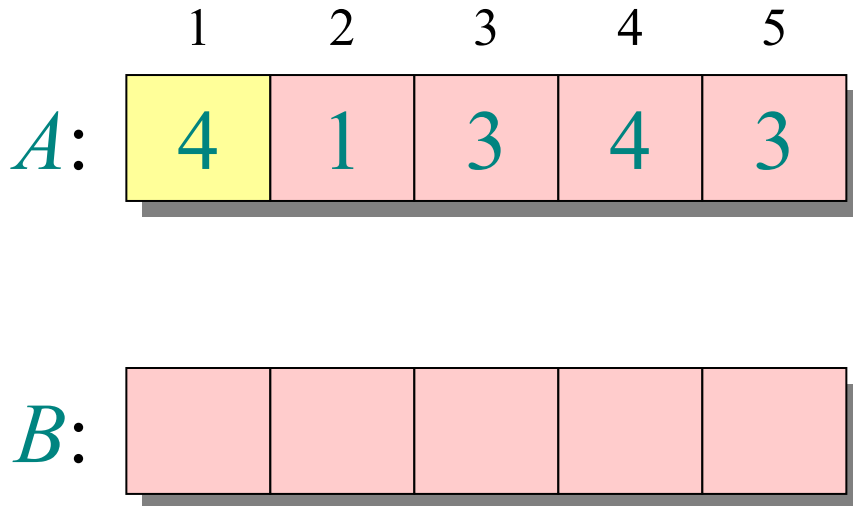
	1	2	3	4
C :	0	0	0	0

B :					
-------	--	--	--	--	--

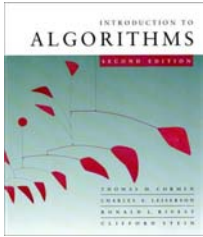
for $i \leftarrow 1$ **to** k
 do $C[i] \leftarrow 0$



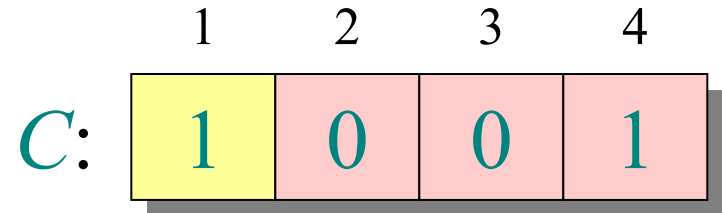
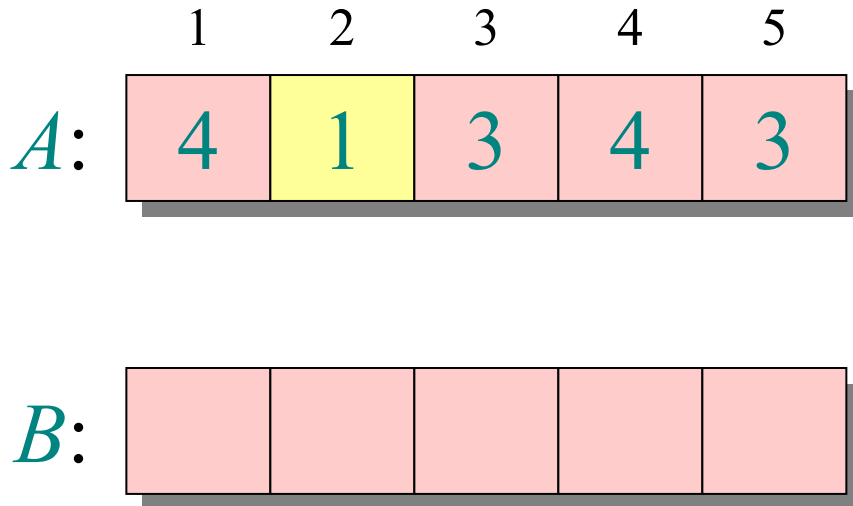
Loop 2



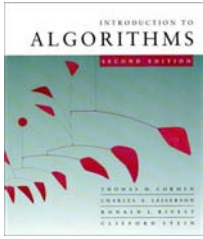
for $j \leftarrow 1$ **to** n
 do $C[A[j]] \leftarrow C[A[j]] + 1$ $\triangleleft C[i] = |\{\text{key} = i\}|$



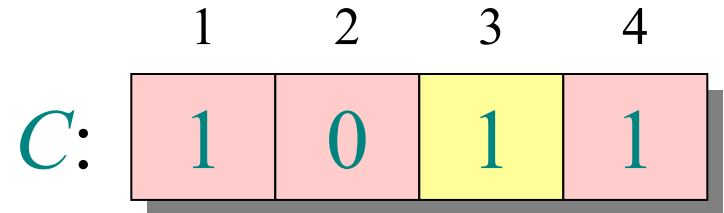
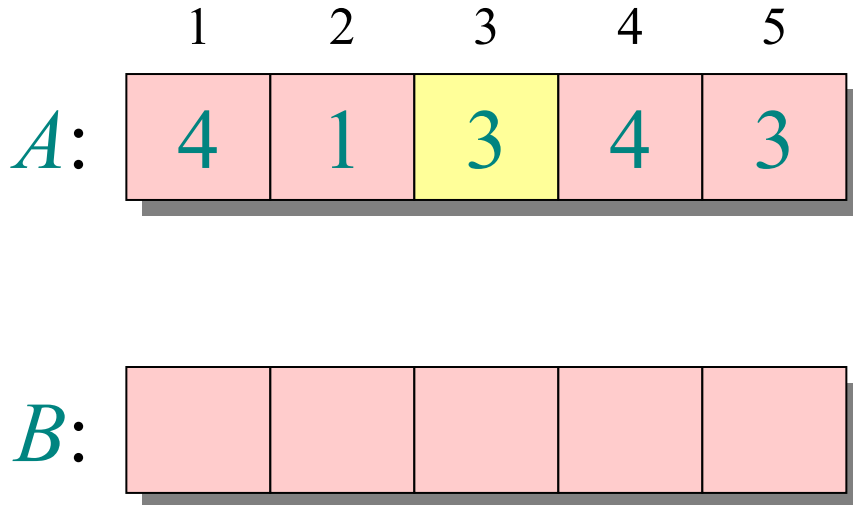
Loop 2



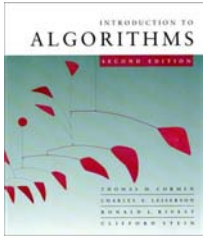
for $j \leftarrow 1$ **to** n
 do $C[A[j]] \leftarrow C[A[j]] + 1$ $\triangleleft C[i] = |\{\text{key} = i\}|$



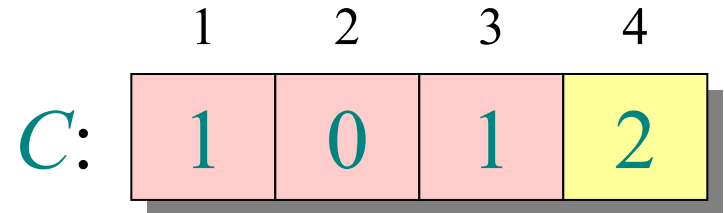
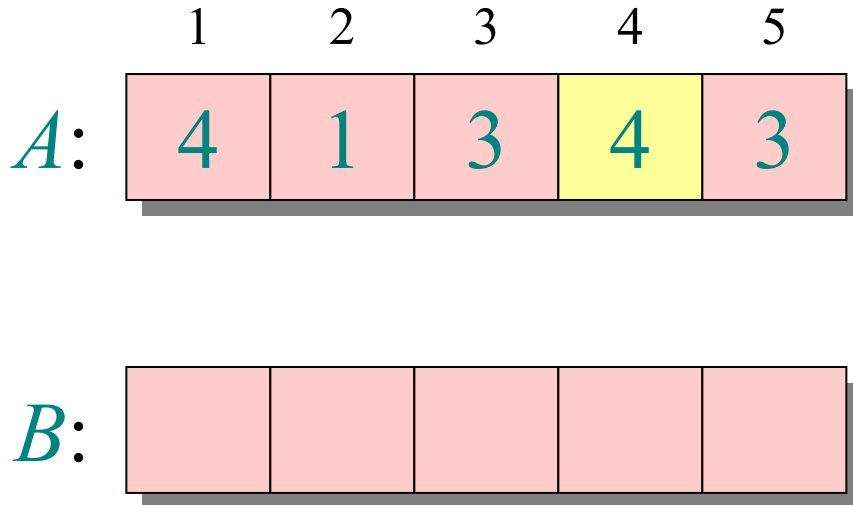
Loop 2



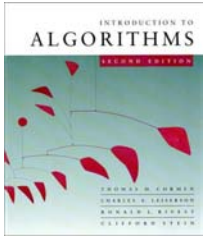
for $j \leftarrow 1$ **to** n
 do $C[A[j]] \leftarrow C[A[j]] + 1$ $\triangleleft C[i] = |\{\text{key} = i\}|$



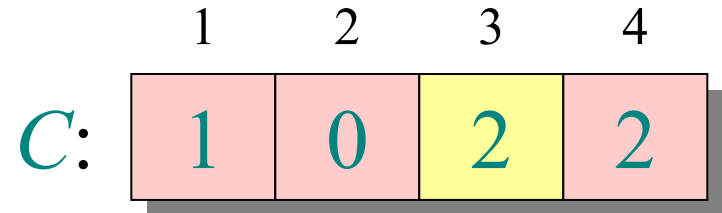
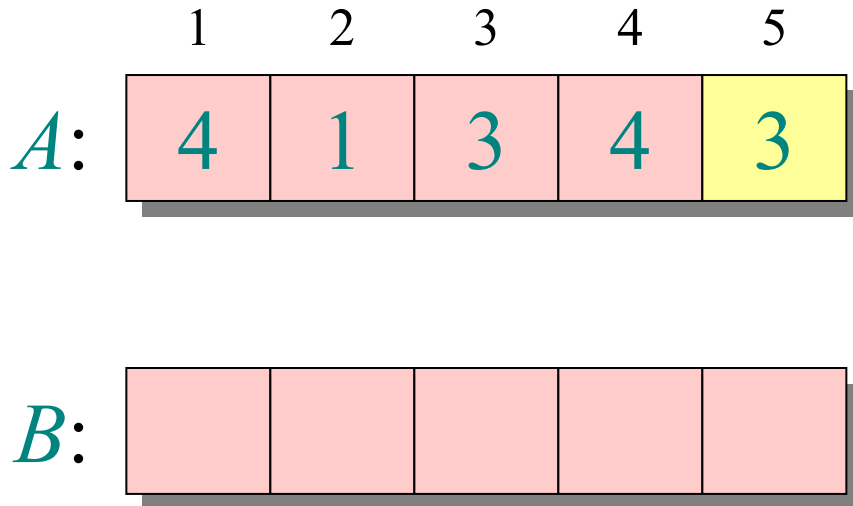
Loop 2



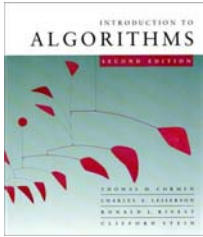
for $j \leftarrow 1$ **to** n
 do $C[A[j]] \leftarrow C[A[j]] + 1$ $\triangleleft C[i] = |\{\text{key} = i\}|$



Loop 2



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



Loop 3

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

B :					
-------	--	--	--	--	--

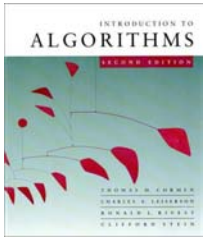
	1	2	3	4
C :	1	0	2	2

C' :	1	1	2	2
--------	---	---	---	---

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

do $C[i] \leftarrow C[i] + C[i-1]$

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



Loop 3

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

	1	2	3	4
C :	1	0	2	2

B :					
-------	--	--	--	--	--

C' :	1	1	3	2
--------	---	---	---	---

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

do $C[i] \leftarrow C[i] + C[i-1]$

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



Loop 3

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

	1	2	3	4
C :	1	0	2	2

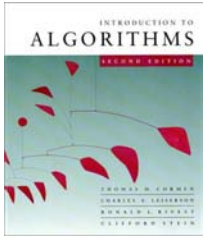
B :					
-------	--	--	--	--	--

C' :	1	1	3	5
--------	---	---	---	---

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

do $C[i] \leftarrow C[i] + C[i-1]$

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



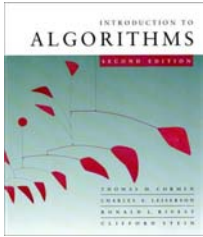
Loop 4

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

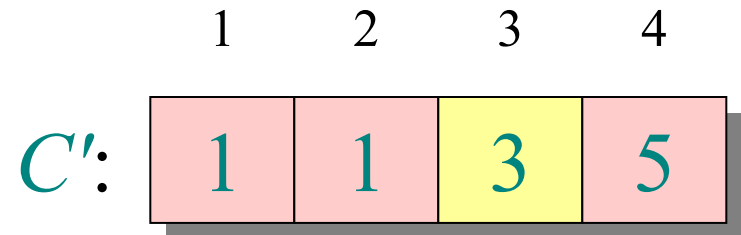
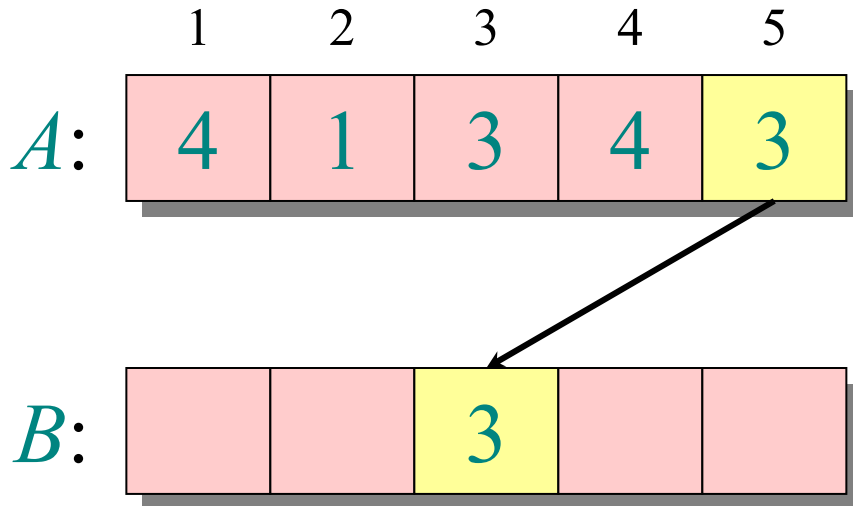
B :					
-------	--	--	--	--	--

	1	2	3	4
C' :	1	1	3	5

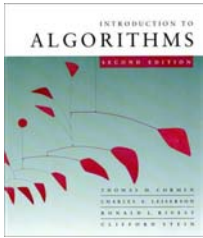
```
for  $j \leftarrow n$  downto 1
  do  $B[C[A[j]]] \leftarrow A[j]$ 
      $C[A[j]] \leftarrow C[A[j]] - 1$ 
```



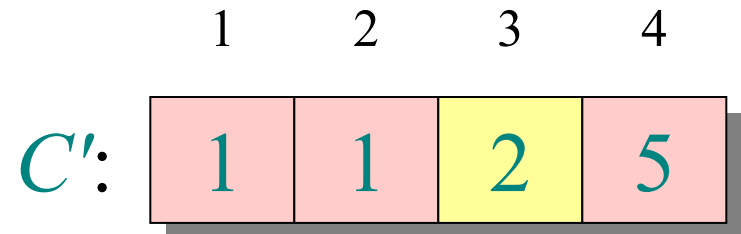
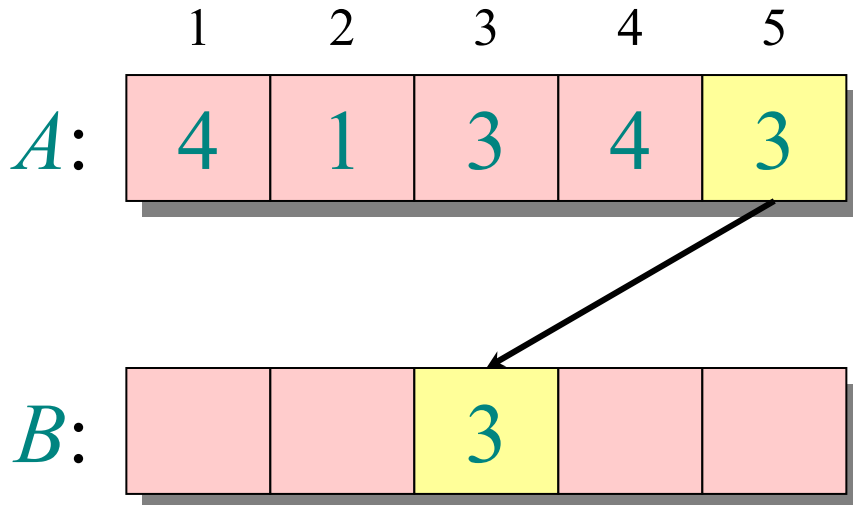
Loop 4



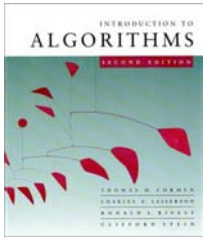
```
for  $j \leftarrow n$  downto 1
  do  $B[C[A[j]]] \leftarrow A[j]$ 
      $C[A[j]] \leftarrow C[A[j]] - 1$ 
```



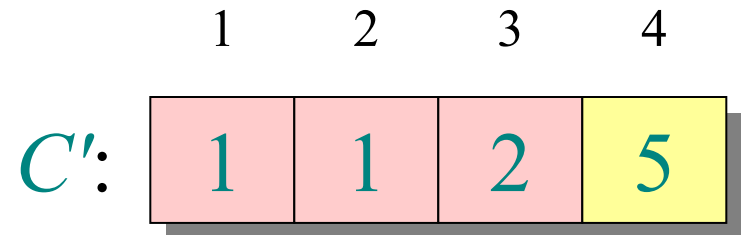
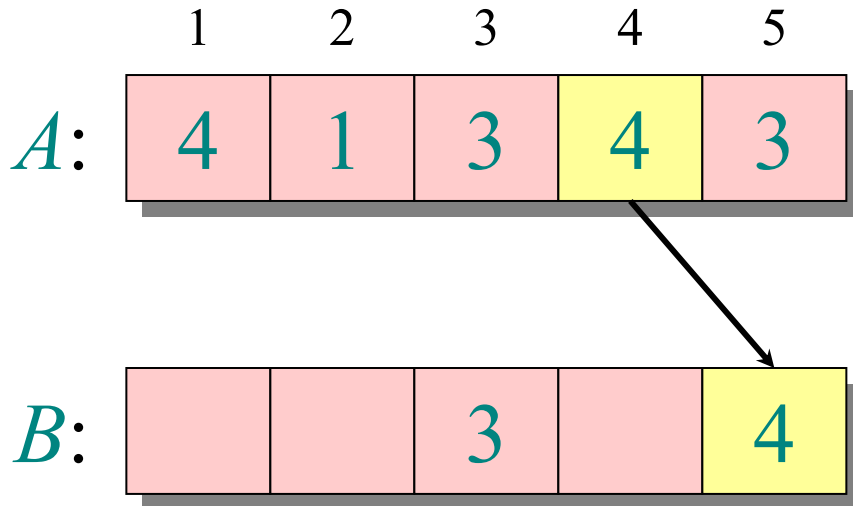
Loop 4



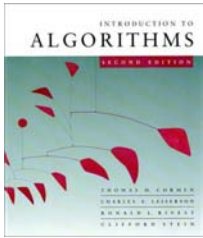
```
for  $j \leftarrow n$  downto 1
  do  $B[C[A[j]]] \leftarrow A[j]$ 
      $C[A[j]] \leftarrow C[A[j]] - 1$ 
```

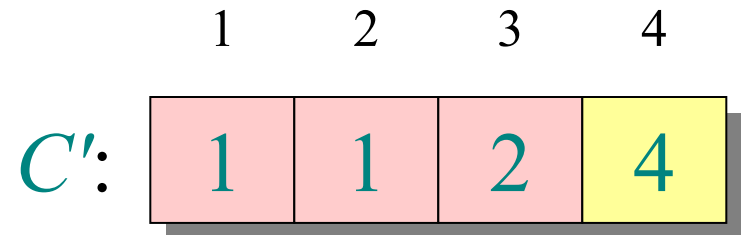
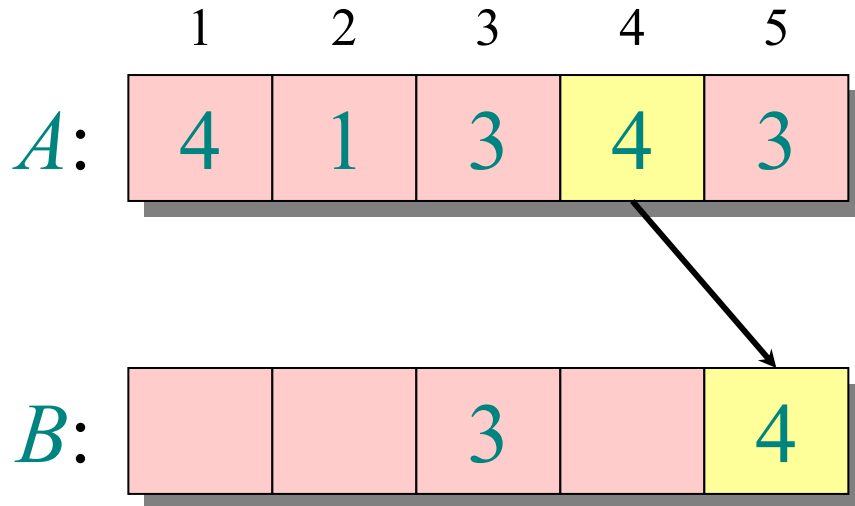
Loop 4



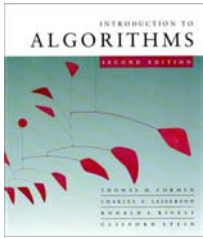
```
for  $j \leftarrow n$  downto 1
  do  $B[C[A[j]]] \leftarrow A[j]$ 
      $C[A[j]] \leftarrow C[A[j]] - 1$ 
```



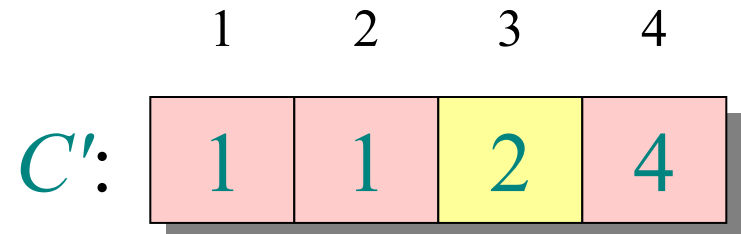
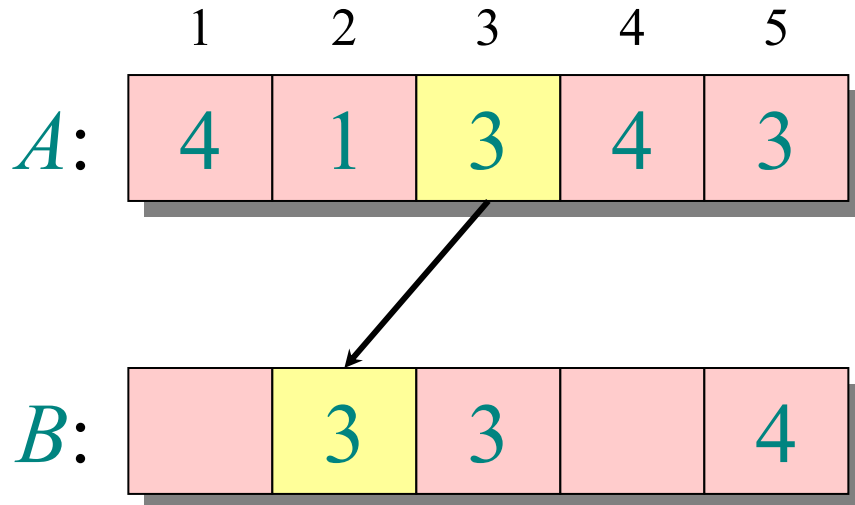
Loop 4



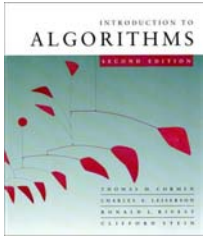
```
for  $j \leftarrow n$  downto 1
  do  $B[C[A[j]]] \leftarrow A[j]$ 
      $C[A[j]] \leftarrow C[A[j]] - 1$ 
```



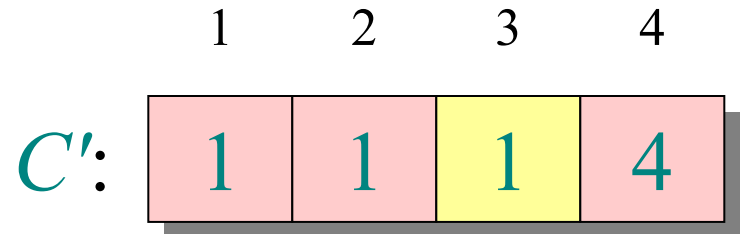
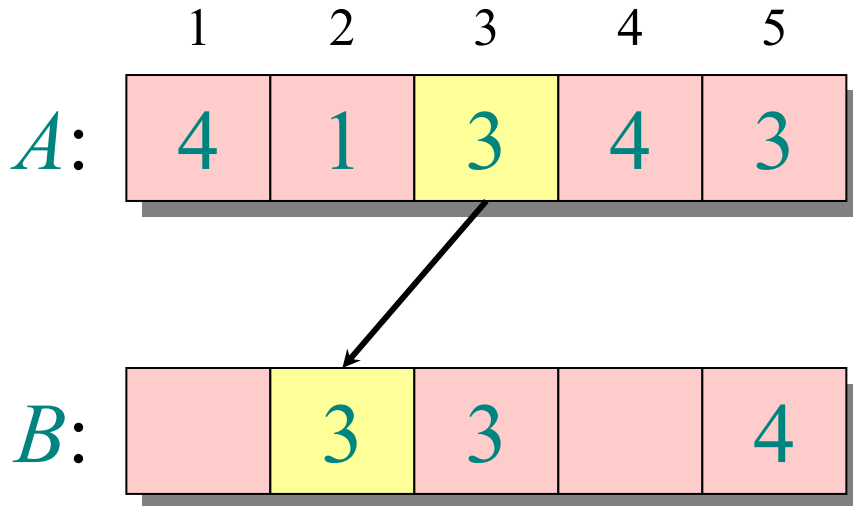
Loop 4



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



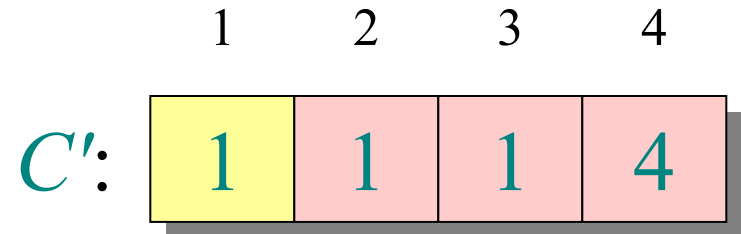
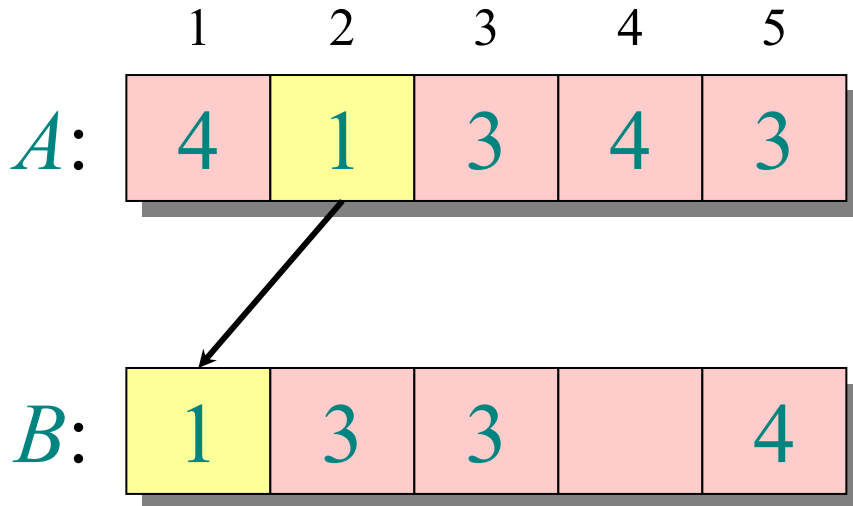
Loop 4



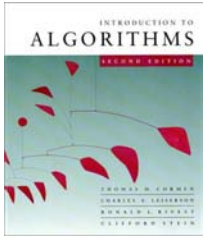
```
for  $j \leftarrow n$  downto 1
  do  $B[C[A[j]]] \leftarrow A[j]$ 
      $C[A[j]] \leftarrow C[A[j]] - 1$ 
```



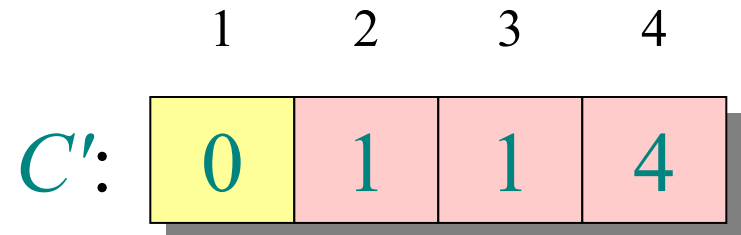
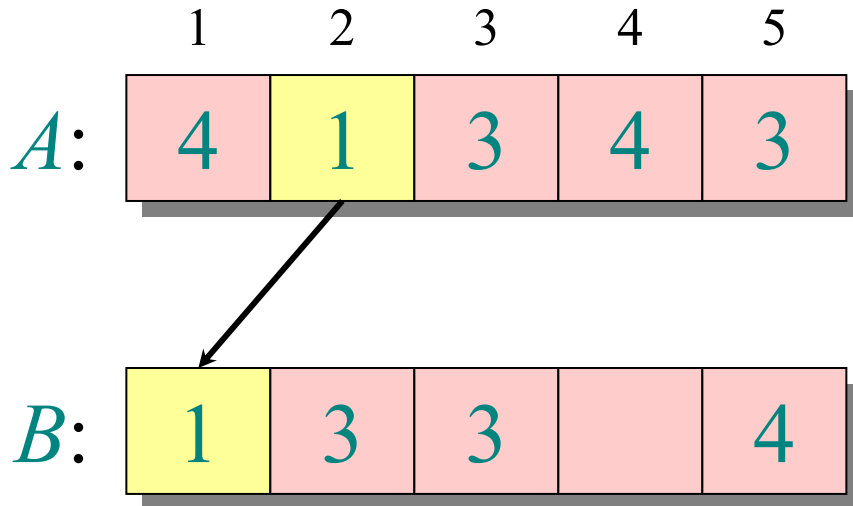
Loop 4



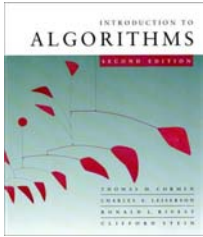
```
for  $j \leftarrow n$  downto 1
  do  $B[C[A[j]]] \leftarrow A[j]$ 
      $C[A[j]] \leftarrow C[A[j]] - 1$ 
```



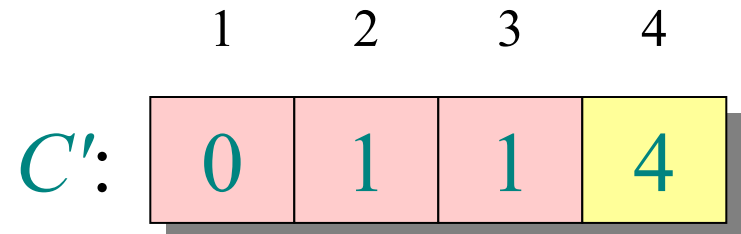
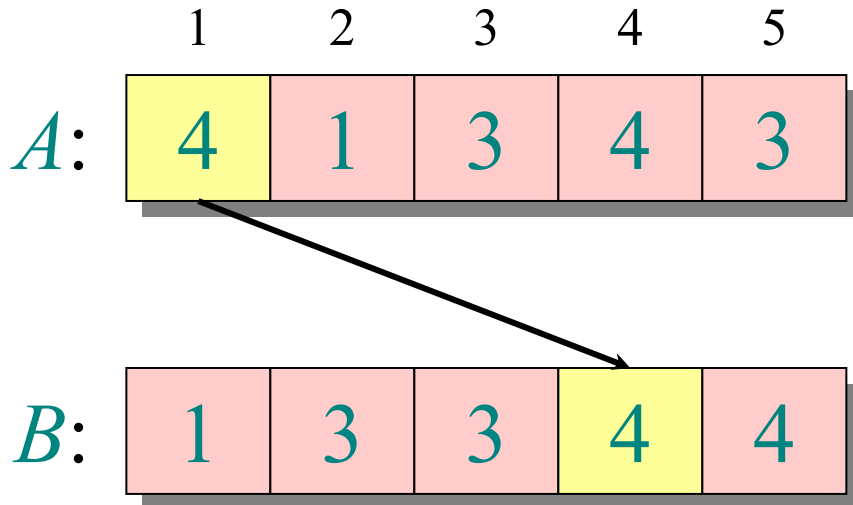
Loop 4



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



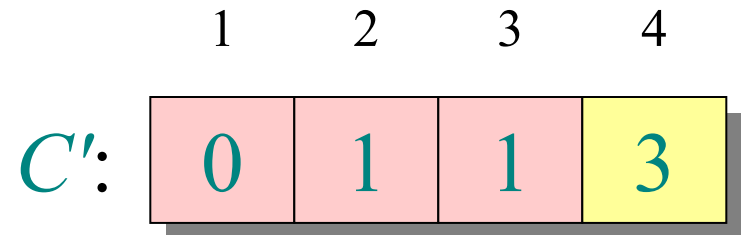
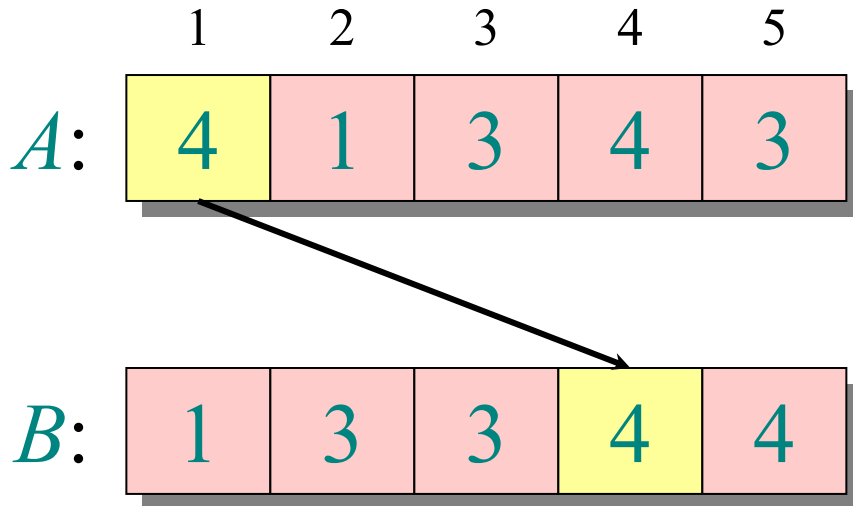
Loop 4



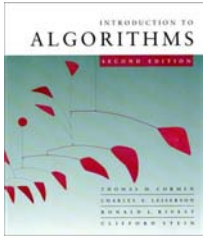
```
for  $j \leftarrow n$  downto 1
  do  $B[C[A[j]]] \leftarrow A[j]$ 
      $C[A[j]] \leftarrow C[A[j]] - 1$ 
```



Loop 4



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

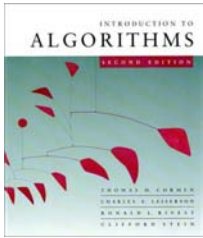



B vs C

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

	1	2	3	4
C' :	1	1	3	5

In the end, each element i occupies the range
 $B[C[i-1]+1 \dots C[i]]$



Analysis

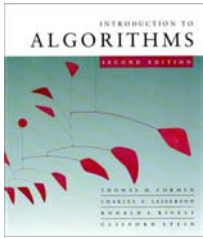
$\Theta(k)$ { **for** $i \leftarrow 1$ **to** k
 do $C[i] \leftarrow 0$

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

$\Theta(n + k)$



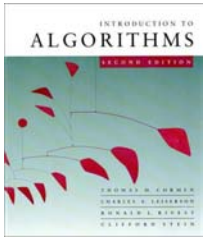
Running time

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

- But, sorting takes $\Omega(n \lg n)$ time!
- Why ?

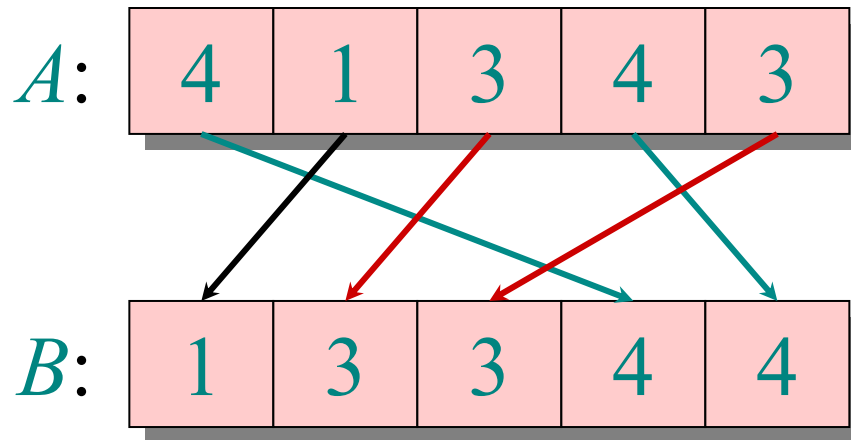
Answer:

- *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

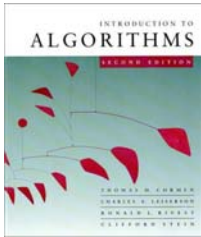
Counting sort is a *stable* sort: it preserves the input order among equal elements.






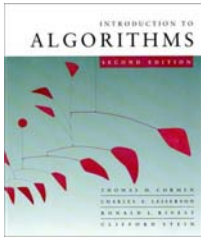
Sorting integers

- We can sort n integers from $\{1, 2, \dots, k\}$ in $O(n+k)$ time
- This is nice if $k=O(n)$
- What if, say, $k=n^2$?

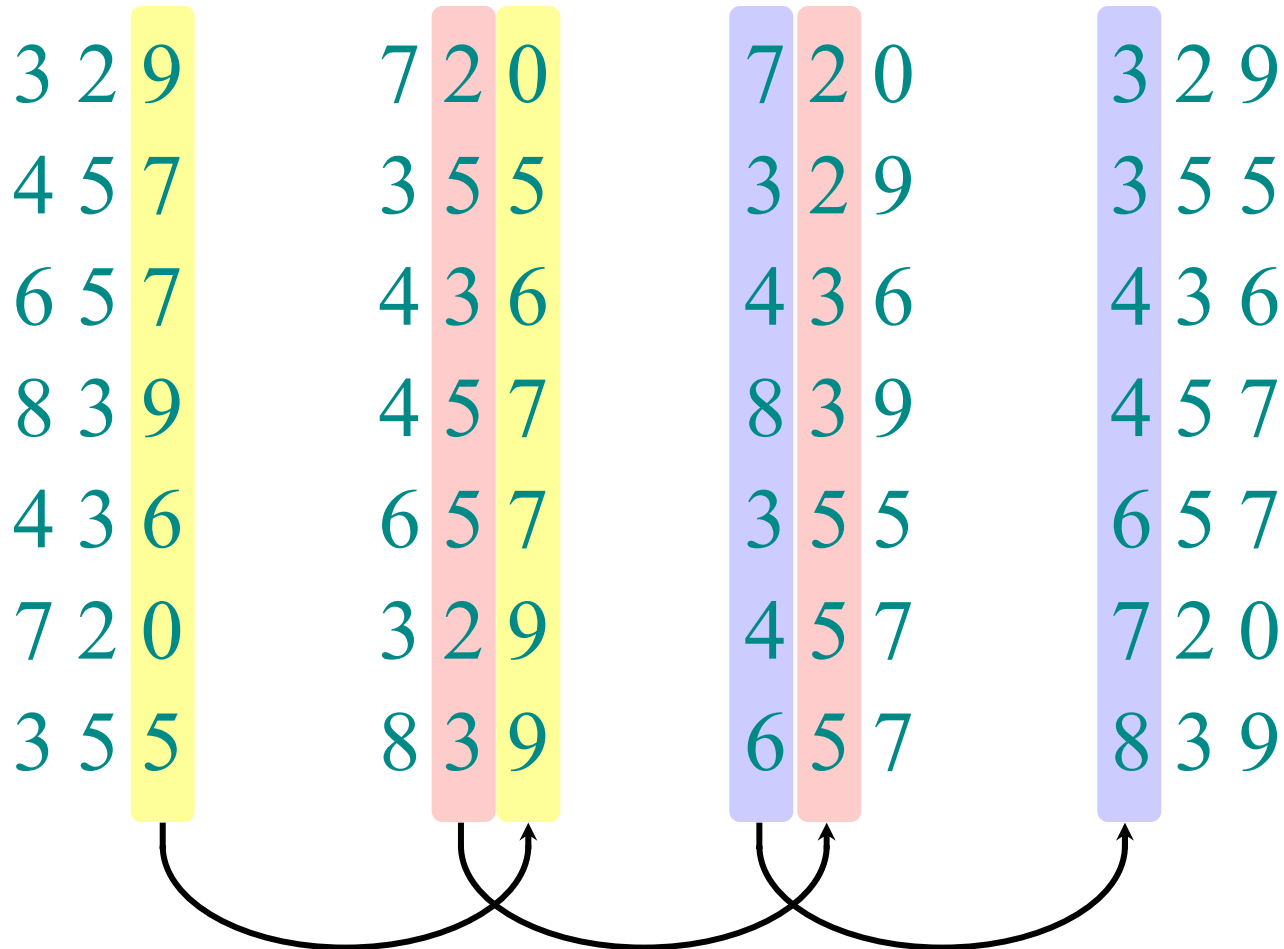


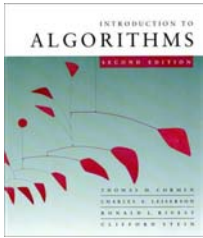
Radix sort

- ***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

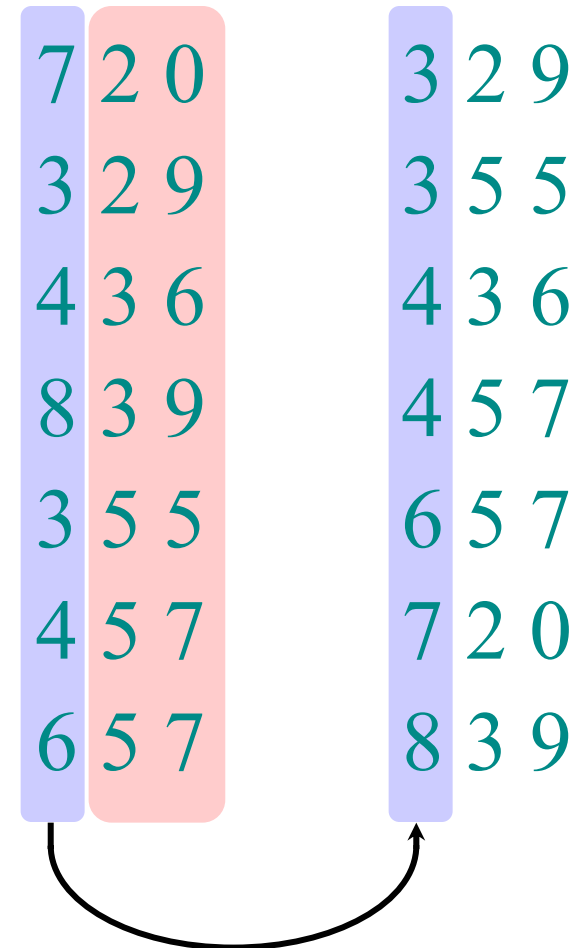


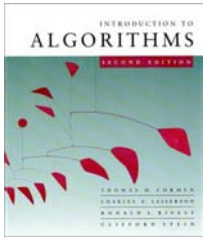


Correctness of radix sort

Induction on digit position

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

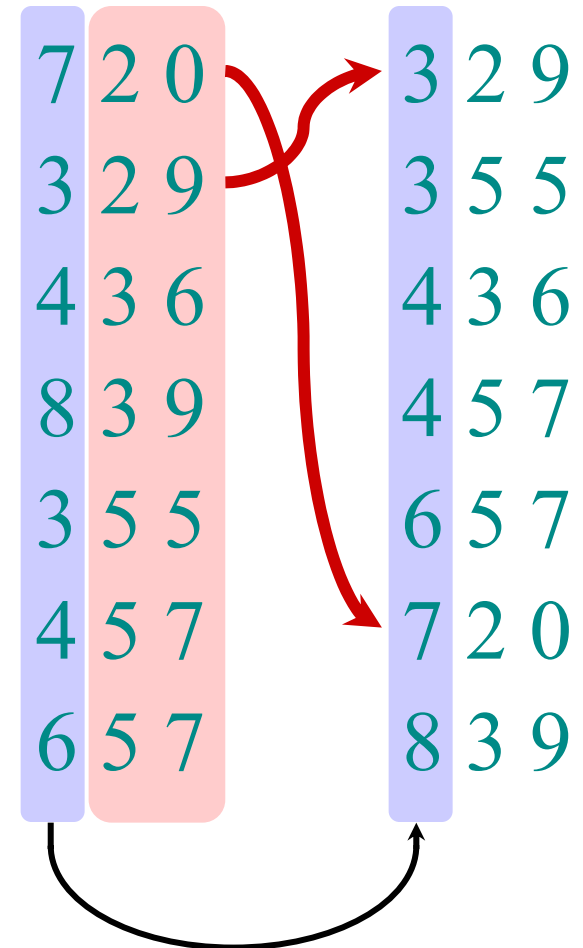


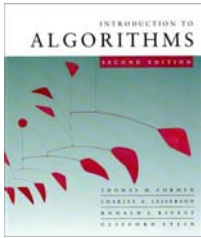


Correctness of radix sort

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.

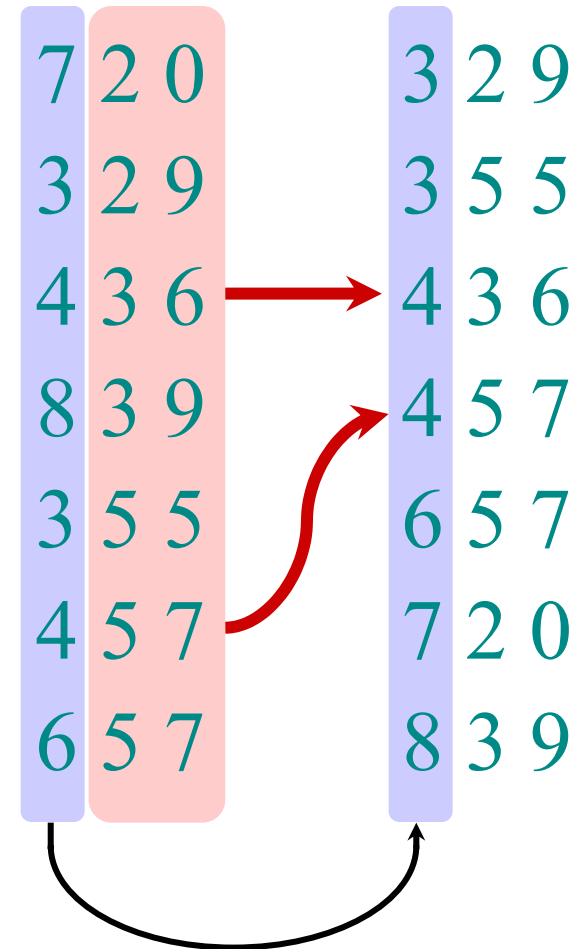


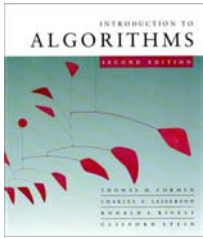


Correctness of radix sort

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.





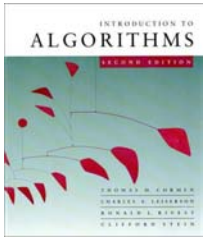
Analysis of radix sort

- Assume counting sort is the auxiliary stable sort.
- Sort n computer words of b bits each
E.g., if we sort elements in $\{1 \dots n^2\}$, $b = 2 \lg n$
- Each word can be viewed as having b/r base- 2^r digits.

Example: 32-bit word

8	8	8	8
---	---	---	---

$r = 8 \Rightarrow b/r = 4$ passes of counting sort on base- 2^8 digits;
or $r = 16 \Rightarrow b/r = 2$ passes of counting sort on base- 2^{16} digits.



Analysis (continued)

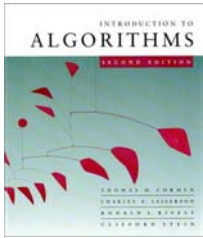
Recall: Counting sort takes $\Theta(n + k)$ time to sort n numbers in the range from 0 to $k - 1$.

If each b -bit word is broken into r -bit pieces, each pass of counting sort takes $\Theta(n + 2^r)$ time. Since there are b/r passes, we have

$$T(n, b) = \Theta\left(\frac{b}{r}(n + 2^r)\right).$$

Choose r to minimize $T(n, b)$:

- Increasing r means fewer passes, but as $r \gg \lg n$, the time grows exponentially.



Choosing r

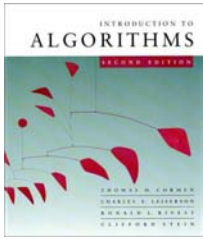
$$T(n, b) = \Theta\left(\frac{b}{r} \left(n + 2^r\right)\right)$$

Minimize $T(n, b)$ by differentiating and setting to 0.

Or, just observe that we don't want $2^r \gg n$, and there's no harm asymptotically in choosing r as large as possible subject to this constraint.

Choosing $r = \lg n$ implies $T(n, b) = \Theta(bn/\lg n)$.

- For numbers in the range from 0 to $n^d - 1$, we have $b = d \lg n \Rightarrow$ radix sort runs in $\Theta(dn)$ time.



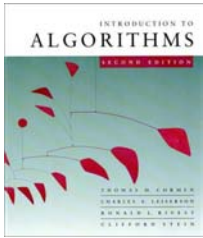
Conclusions

In practice, radix sort is fast for large inputs, as well as simple to code and maintain.

Example (32-bit numbers):

- At most 3 passes when sorting ≥ 2000 numbers.
- Merge sort and quicksort do at least $\lceil \lg 2000 \rceil = 11$ passes.

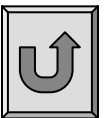
Downside: Unlike quicksort, radix sort displays little locality of reference.

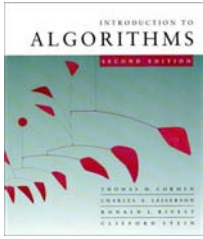


Appendix: Punched-card technology

- 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

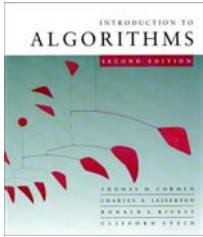
Return to last
slide viewed.





Herman Hollerith (1860-1929)

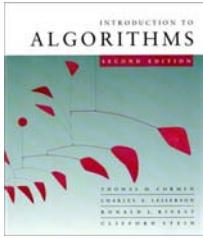
- The 1880 U.S. Census took almost 10 years to process.
Image removed due to copyright considerations.
- While a lecturer at MIT, Hollerith prototyped punched-card technology.
- His machines, including a “card sorter,” allowed the 1890 census total to be reported in 6 weeks.
- He founded the Tabulating 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.

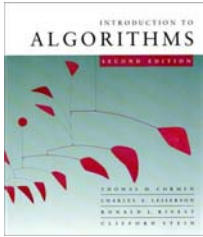
Image removed due to copyright considerations.



Hollerith's tabulating system

- Pantograph card punch
- Hand-press reader
- Dial counters
- Sorting box

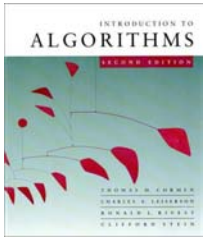
Image removed due to copyright considerations.



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.

Image removed due to copyright considerations.



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.

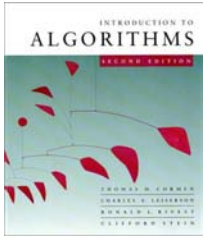


“Modern” IBM card

- One character per column.

Image removed due to copyright considerations.

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](#)