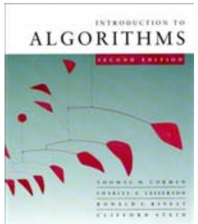
CS 3343 – Fall 2007



Sorting

Carola Wenk

Slides courtesy of Charles Leiserson with small changes by Carola Wenk

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.

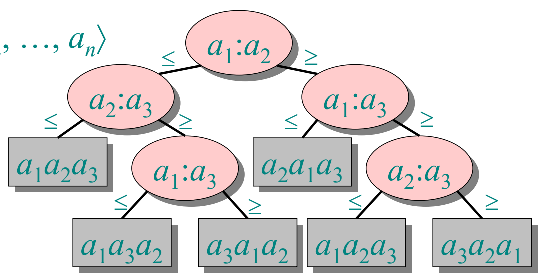
The best worst-case running time that we've seen for comparison sorting is $O(n \log n)$.

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

Decision trees can help us answer this question.

Decision-tree example

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

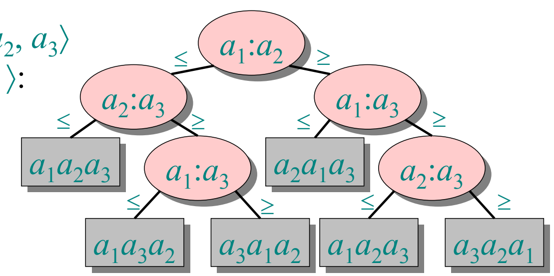


Each internal node is labeled $a_i:a_j$ for $i, j \in \{1, 2, \ldots, n\}$.

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

Decision-tree example

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

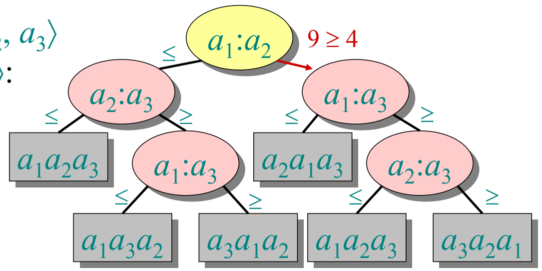


Each internal node is labeled $a_i:a_j$ for $i, j \in \{1, 2, \ldots, n\}$.

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

Decision-tree example

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

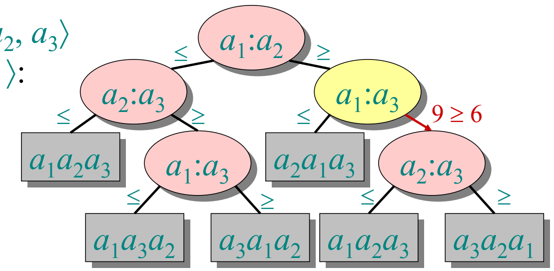


Each internal node is labeled $a_i{:}a_j$ for $i, j \in \{1, 2, \ldots, n\}$.

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

Decision-tree example

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

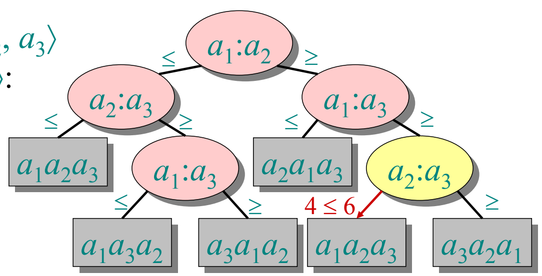


Each internal node is labeled $a_i{:}a_j$ for $i, j \in \{1, 2, \ldots, n\}$.

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

Decision-tree example

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

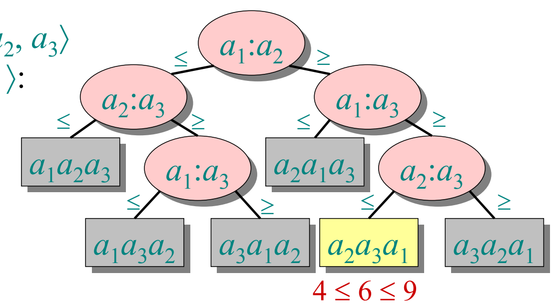


Each internal node is labeled $a_i{:}a_j$ for $i, j \in \{1, 2, \ldots, n\}$.

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

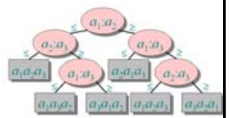
Decision-tree example

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 $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$ has been established.

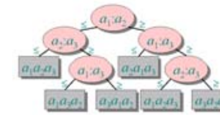
Decision-tree model



A decision tree can model the execution of any comparison sorting algorithm:

- One tree for each input size n.
- The tree contains **all** possible comparisons (= if-branches) that could be executed for **any** input of size n.
- The tree contains **all** comparisons along **all** possible instruction traces (= control flows) for **all** inputs of size n.
- For one input, only one path to a leaf is executed.
- Running time = length of the path taken.
- Worst-case running time = height of tree.

Lower bound for comparison 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, $n! \leq 2^h$.

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

Lower bound for comparison sorting

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

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

1.for $i \leftarrow 1$ **to** k
    do $C[i] \leftarrow 0$
2.for $j \leftarrow 1$ **to** n
    do $C[A[j]] \leftarrow C[A[j]] + 1$ $\triangleright$ $C[i] = |\{\text{key} = i\}|$
3.for $i \leftarrow 2$ **to** k
    do $C[i] \leftarrow C[i] + C[i{-}1]$ $\triangleright$ $C[i] = |\{\text{key} \le i\}|$
4.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

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

	1	2	3	4
C:				

Loop 1

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

	1	2	3	4
C:	0	0	0	0

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

Loop 2

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

	1	2	3	4
C:	0	0	0	1

2.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	0	1

B:

2.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	1

B:

2.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:

2.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:

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

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

3. for $i \leftarrow 2$ **to** k
 do $C[i] \leftarrow C[i] + C[i–1]$ ▷ $C[i] = |\{\text{key} \leq i\}|$

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	3	2

3. for $i \leftarrow 2$ **to** k
 do $C[i] \leftarrow C[i] + C[i–1]$ ▷ $C[i] = |\{\text{key} \leq i\}|$

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	3	5

3. for $i \leftarrow 2$ **to** k
 do $C[i] \leftarrow C[i] + C[i–1]$ ▷ $C[i] = |\{\text{key} \leq i\}|$

Loop 4

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

	1	2	3	4
C:	1	1	3	5
C′:	1	1	3	5

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

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

	1	2	3	4
C:	1	1	3	5
C′:	1	1	2	5

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

	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	5

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

	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

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

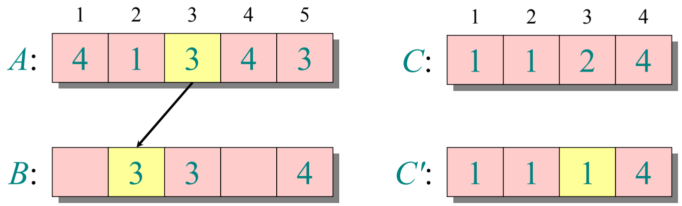
Loop 4

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

	1	2	3	4
C:	1	1	2	4
C′:	1	1	2	4

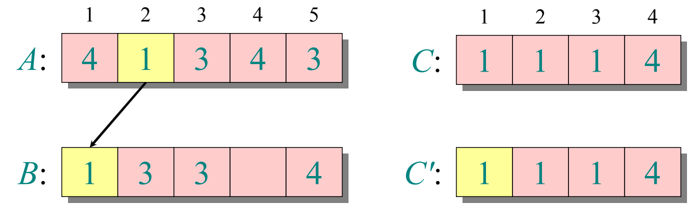
4. for $j \leftarrow n$ **downto** 1
  do $B[C[A[j]]] \leftarrow A[j]$
    $C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4



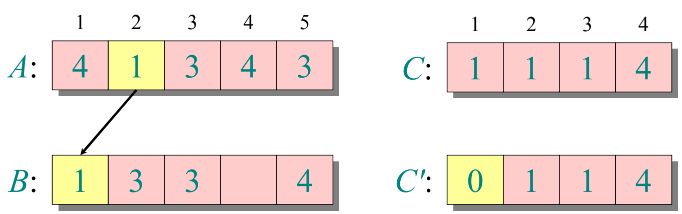
4. **for** $j \leftarrow n$ **downto** 1
   do $B[C[A[j]]] \leftarrow A[j]$
   $C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4



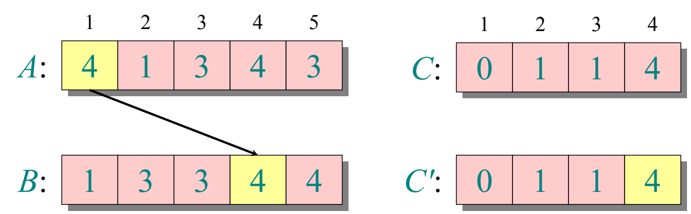
4. **for** $j \leftarrow n$ **downto** 1
   do $B[C[A[j]]] \leftarrow A[j]$
   $C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4



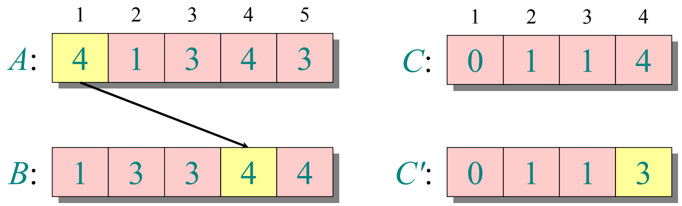
4. **for** $j \leftarrow n$ **downto** 1
   do $B[C[A[j]]] \leftarrow A[j]$
   $C[A[j]] \leftarrow C[A[j]] - 1$

Loop 4



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



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

Analysis

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

$\Theta(n)$ — **2.for** $j \leftarrow 1$ **to** n
  do $C[A[j]] \leftarrow C[A[j]] + 1$

$\Theta(k)$ — **3.for** $i \leftarrow 2$ **to** k
  do $C[i] \leftarrow C[i] + C[i-1]$

$\Theta(n)$ — **4.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.

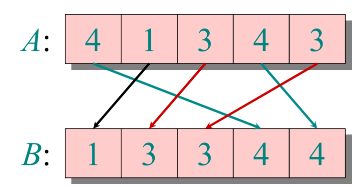
- But, sorting takes $\Omega(n \log n)$ time!
- Where's the fallacy?

Answer:

- ***Comparison sorting*** takes $\Omega(n \log n)$ 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?