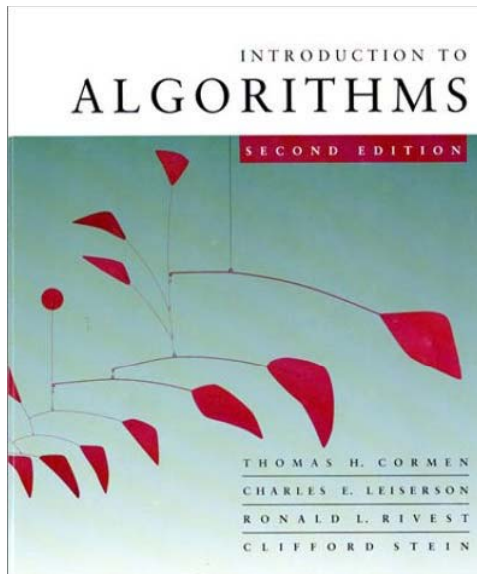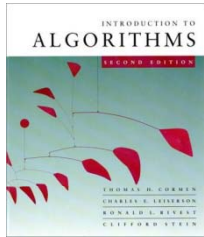# CS 5633 -- Spring 2012

Augmenting Data Structures

Carola Wenk

Slides courtesy of Charles Leiserson with small changes by Carola Wenk
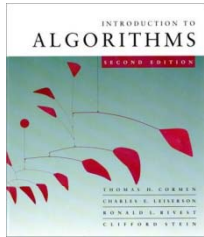
Dictionaries and Dynamic Sets

Abstract Data Type (ADT) Dictionary :

Insert (x, D):   inserts x into D

Delete (x, D):   deletes x from D D is a dynamic set

Find (x, D):   finds x in D

Popular implementation uses any balanced search tree (not necessarily binary). This way each operation takes $O(\log n)$ time.
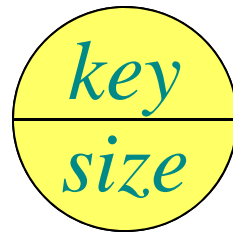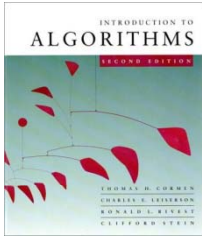
Dynamic order statistics

OS-SELECT(i, S):   returns the ith smallest element in the dynamic set S.

OS-RANK(x, S):   returns the rank of $x \in S$ in the sorted order of S's elements.
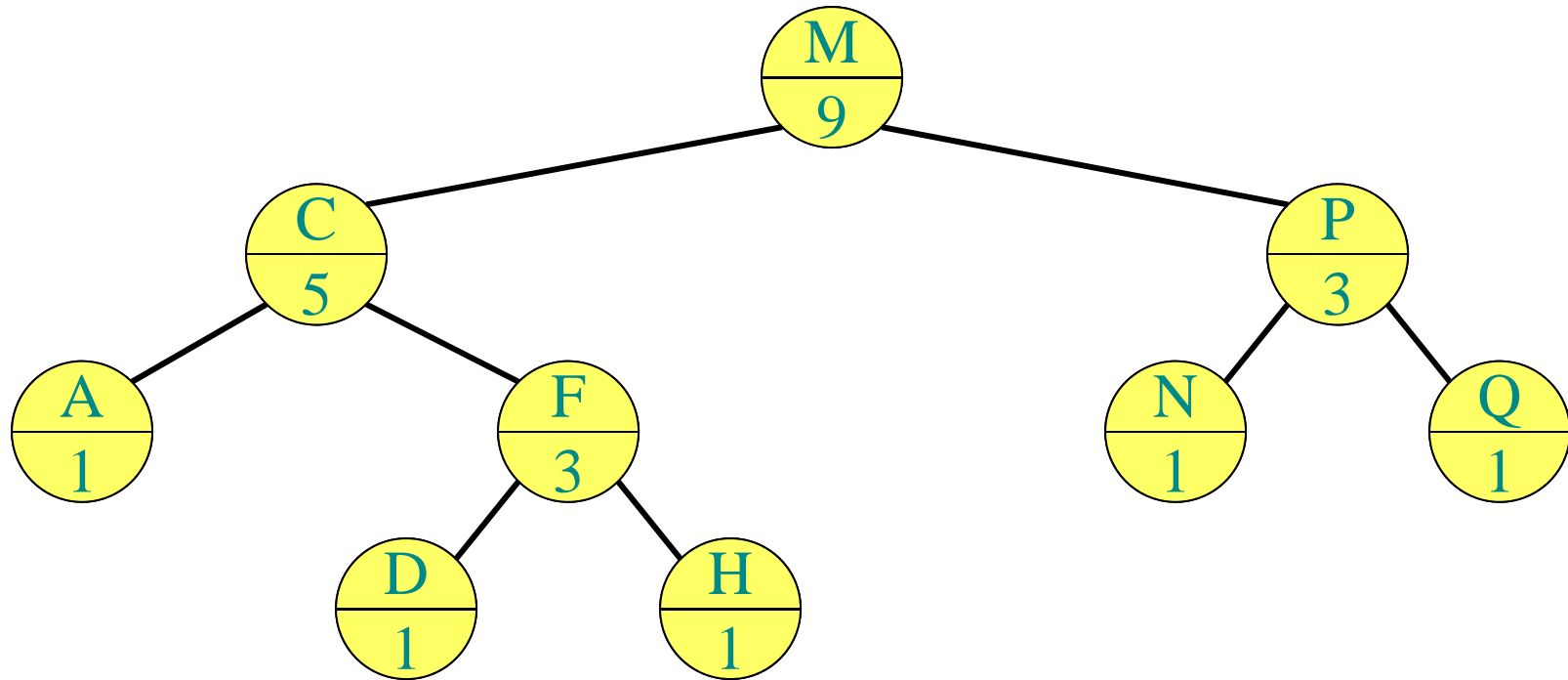
IDEA: Use a red-black tree for the set S, but keep subtree sizes in the nodes.
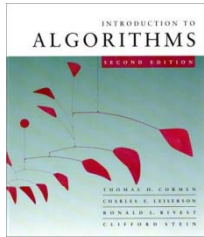
Notation for nodes:

$$\frac{key}{size}$$

Example of an OS-tree

$$size[x] = size[left[x]] + size[right[x]] + 1$$

CS 5633 Analysis of Algorithms

Selection

Implementation trick: Use a *sentinel* (dummy record) for NIL such that $size[\text{NIL}] = 0$.

OS-SELECT(x, i) $\triangleright\, i$th smallest element in the subtree rooted at x

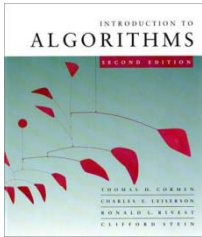$\quad k \leftarrow size[left[x]] + 1 \quad \triangleright\, k = \text{rank}(x)$

$\quad$**if** $i = k$ **then return** x

$\quad$**if** $i < k$

$\qquad$**then return** OS-SELECT$(left[x], i)$

$\qquad$**else return** OS-SELECT$(right[x], i - k)$

(OS-RANK is in the textbook.)

Example

OS-SELECT(x, i) ▷ ith smallest element in the
 subtree rooted at x

$k \leftarrow size[left[x]] + 1$ ▷ $k = \text{rank}(x)$
if $i = k$ **then return** x
if $i < k$
  then return OS-SELECT$(left[x], i)$
  else return OS-SELECT$(right[x], i - k)$

OS-SELECT$(root, 5)$

$i = 5$ $k = 6$

$i = 5$ $k = 2$

$i = 3$ $k = 2$

$i = 1$ $k = 1$

$i = 3$ $k = 2$

$i = 1$ $k = 1$
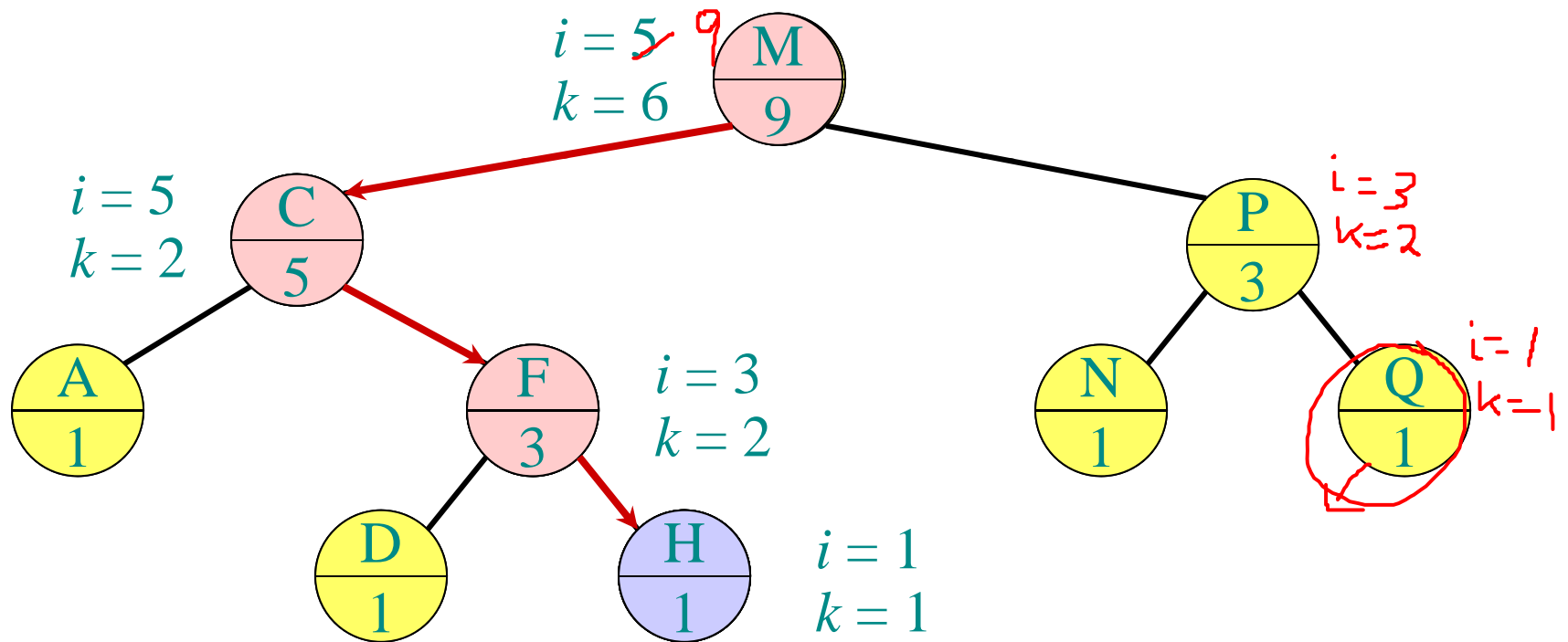
Running time $= O(h) = O(\log n)$ for red-black trees.

CS 5633 Analysis of Algorithms
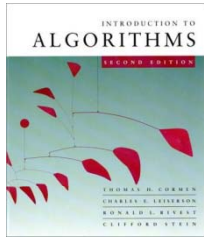
Data structure maintenance

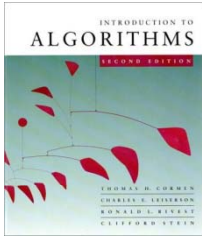**Q.** Why not keep the ranks themselves in the nodes instead of subtree sizes?

A. They are hard to maintain when the red-black tree is modified.

$$k \leftarrow size[left[x]] + 1 \qquad \triangleright\ k = \text{rank}(x)$$

Modifying operations: INSERT and DELETE.

Strategy: Update subtree sizes when inserting or deleting.

Example of insertion

INSERT("K")

CS 5633 Analysis of Algorithms
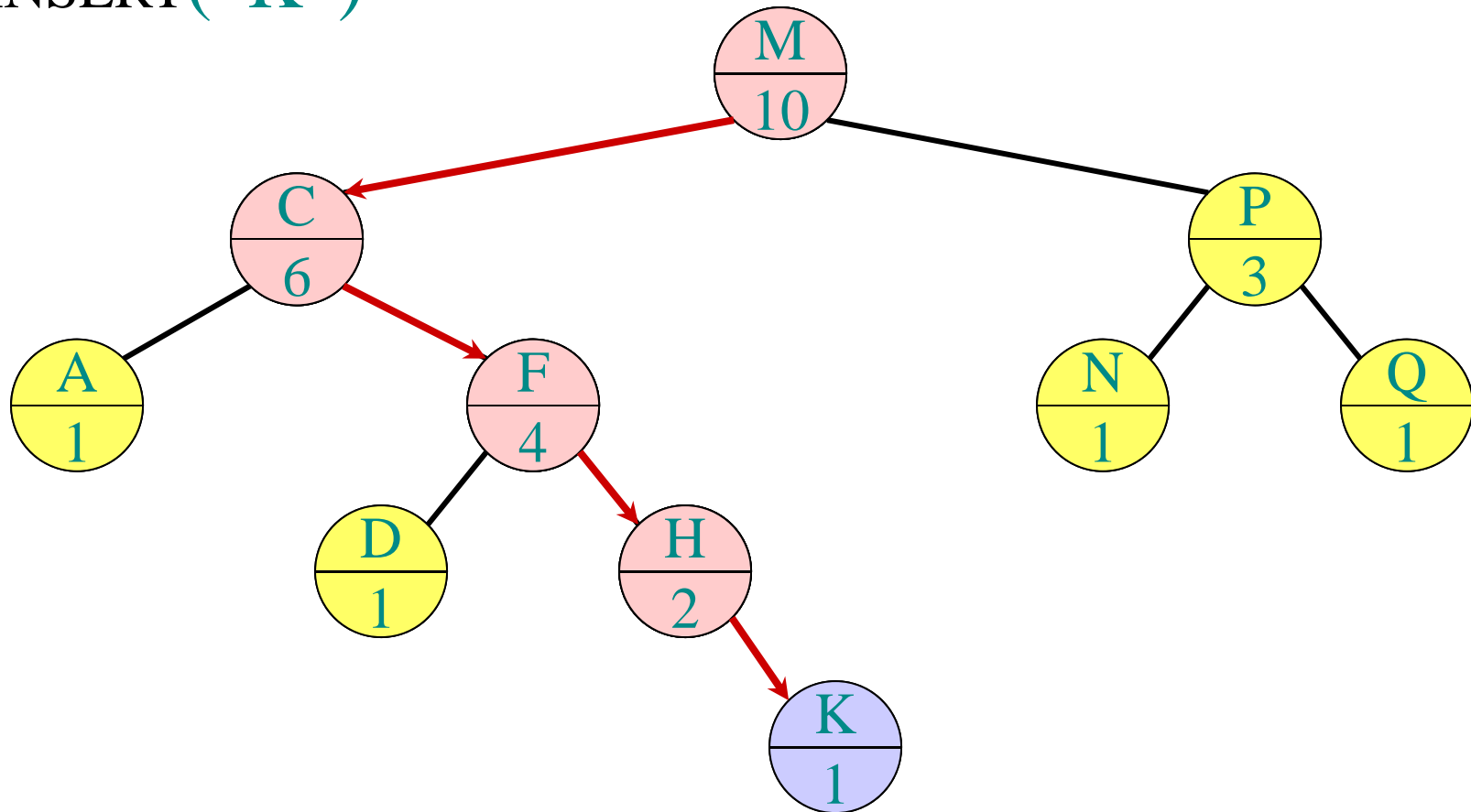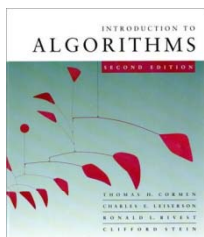
Handling rebalancing

Don't forget that RB-INSERT and RB-DELETE may also need to modify the red-black tree in order to maintain balance.

- *Recolorings*: no effect on subtree sizes.
- *Rotations*: fix up subtree sizes in $O(1)$ time.

Example:
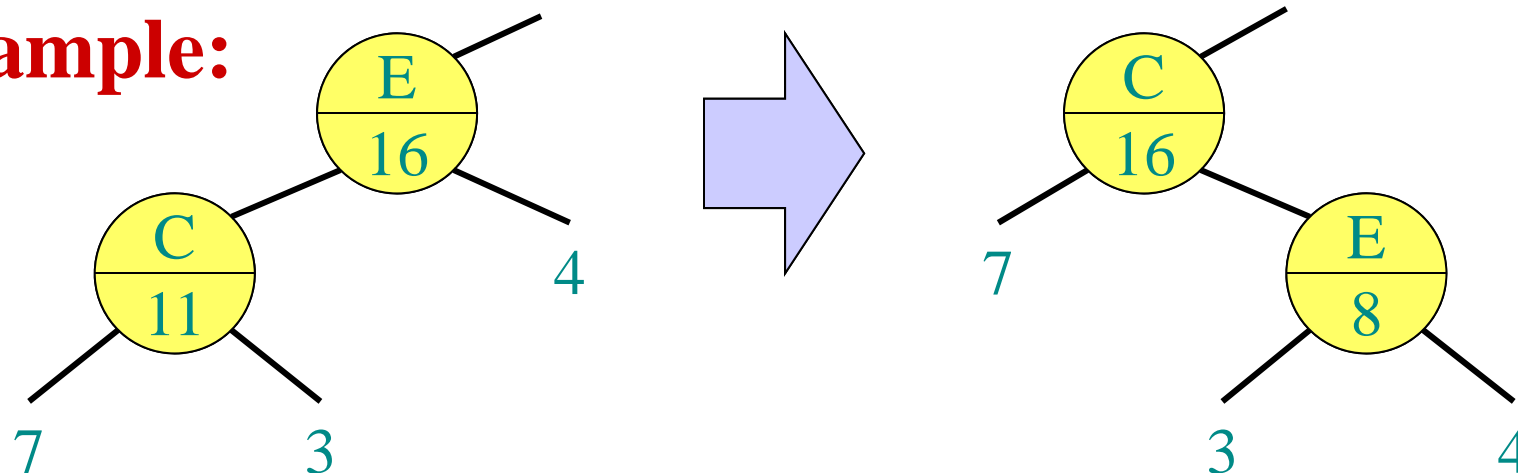
∴RB-INSERT and RB-DELETE still run in $O(\log n)$ time.

Data-structure augmentation

Methodology: (*e.g., order-statistics trees*)

1. Choose an underlying data structure (*red-black tree*).
2. Determine additional information to be stored in the data structure (*subtree sizes*).
3. Verify that this information can be maintained for modifying operations (RB-INSERT, RB-DELETE — *don't forget rotations*).
4. Develop new dynamic-set operations that use the information (OS-SELECT *and* OS-RANK).

These steps are guidelines, not rigid rules.

Interval trees

Goal: To maintain a dynamic set of intervals, such as time intervals.

$$i = [7, 10]$$

$low[i] = 7 \bullet\!\!-\!\!\bullet\ 10 = high[i]$

5 $\bullet\!\!-\!\!\bullet$ 11   17 $\bullet\!\!-\!\!\bullet$ 19

4 $\bullet\!\!-\!\!\bullet$ 8   15 $\bullet\!\!-\!\!\bullet$ 18   22 $\bullet\!\!-\!\!\bullet$ 23
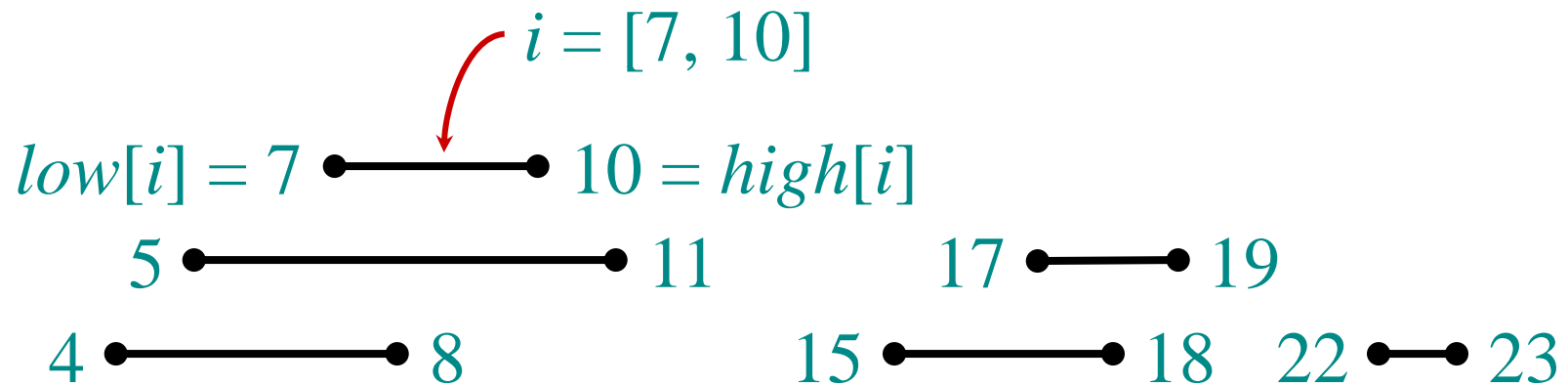
Query: For a given query interval i, find an interval in the set that overlaps i.

CS 5633 Analysis of Algorithms
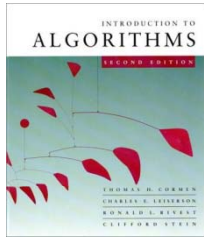
Following the methodology

1. *Choose an underlying data structure.*
   - Red-black tree keyed on low (left) endpoint.
2. *Determine additional information to be stored in the data structure.*
   - Store in each node x the interval $int[x]$ corresponding to the key, as well as the largest value $m[x]$ of all right interval endpoints stored in the subtree rooted at x.

$$\begin{array}{c} int \\ \hline m \end{array}$$

Example interval tree

int
m

$low[i] = 7 \bullet\!\!-\!\!\bullet 10 = high[i]$

$5 \bullet\!\!-\!\!\!-\!\!\!-\!\!\bullet 11 \qquad 17 \bullet\!\!-\!\!\bullet 19$

$4 \bullet\!\!-\!\!\!-\!\!\!-\!\!\bullet 8 \qquad 15 \bullet\!\!-\!\!\!-\!\!\bullet 18 \quad 22 \bullet\!\!-\!\!\bullet 23$

17,19
23

red

5,11
18

22,23
23

4,8
8

15,18
18

7,10
10
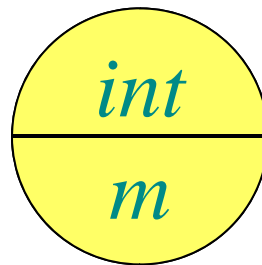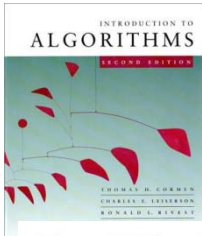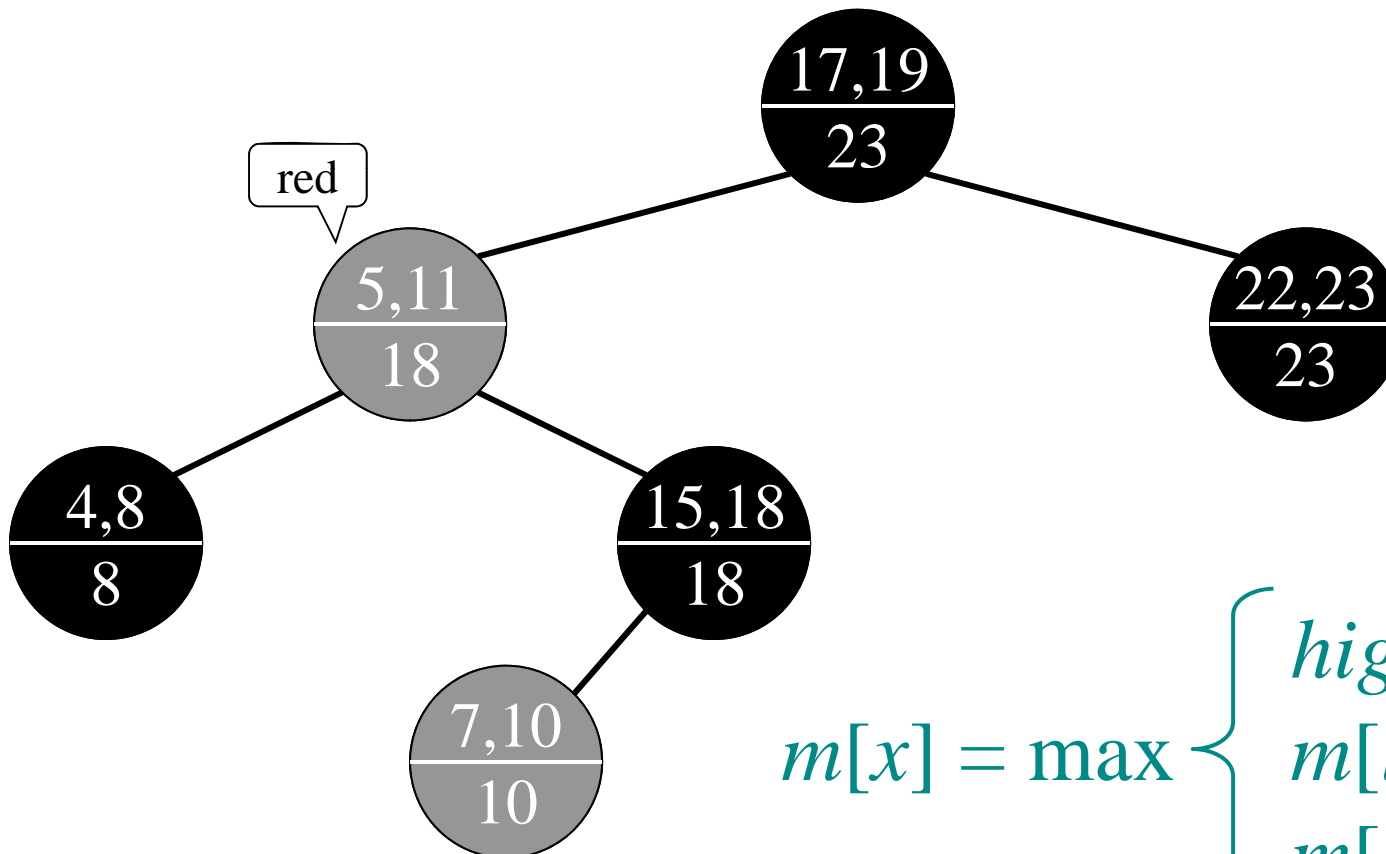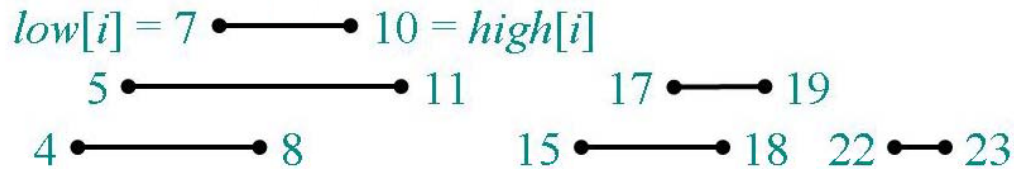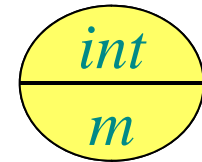
$$m[x] = \max \begin{cases} high[int[x]] \\ m[left[x]] \\ m[right[x]] \end{cases}$$
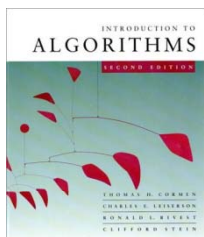
Modifying operations

3. Verify that this information can be maintained for modifying operations.

- INSERT: Fix m's on the way down.
- Rotations — Fixup $= O(1)$ time per rotation:
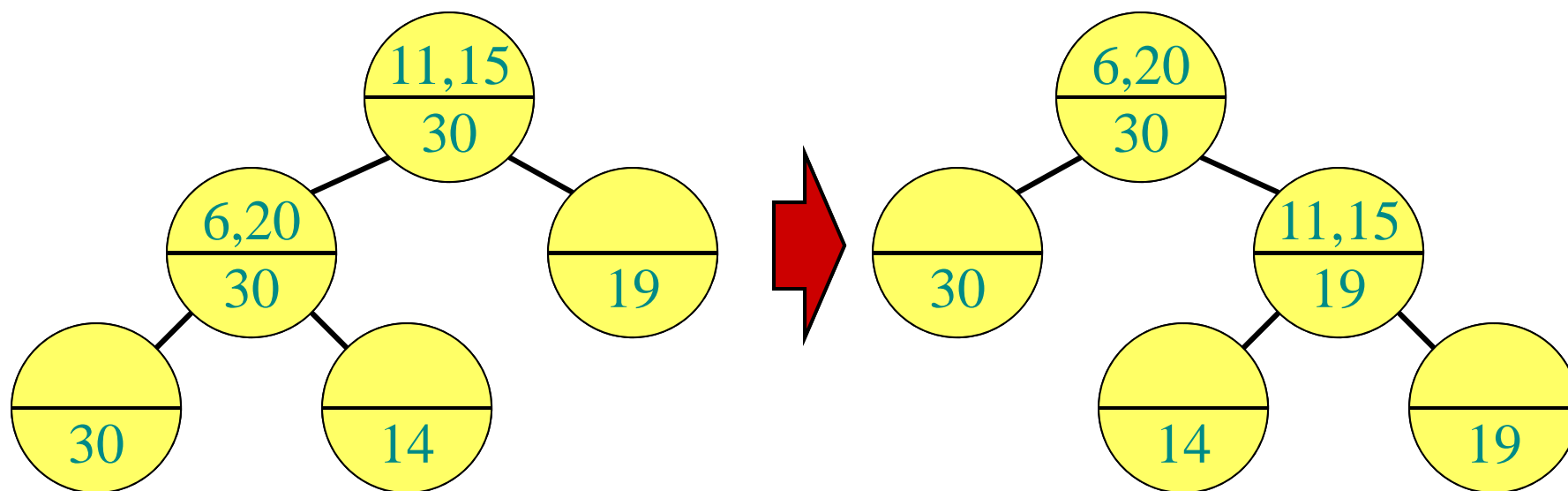
Total INSERT time $= O(\log n)$; DELETE similar.

CS 5633 Analysis of Algorithms

New operations

4. Develop new dynamic-set operations that use the information.

INTERVAL-SEARCH(i)

$x \leftarrow root$

while $x \neq$ NIL **and** ($low[i] > high[int[x]]$

$\qquad\qquad\qquad\qquad$ or $low[int[x]] > high[i]$)

$\quad$ **do** $\triangleright$ i and $int[x]$ don't overlap

$\qquad$ **if** $left[x] \neq$ NIL **and** $low[i] \leq m[left[x]]$

$\qquad\qquad$ **then** $x \leftarrow left[x]$

$\qquad\qquad$ **else** $x \leftarrow right[x]$

return x

CS 5633 Analysis of Algorithms

Example 1: INTERVAL-SEARCH([14,16])


```
low[i] = 7 •——————• 10 = high[i]
      5 •——————————————• 11        17 •———• 19
    4 •——————————• 8              15 •———• 18   22 •——• 23
                                      14 •—• 16
```

while $x \neq$ NIL and $(low[i] > high[int[x]]$
                        or $low[int[x]] > high[i])$
   do ▷ i and $int[x]$ don't overlap
      if $left[x] \neq$ NIL and $low[i] \leq m[left[x]]$
         then $x \leftarrow left[x]$
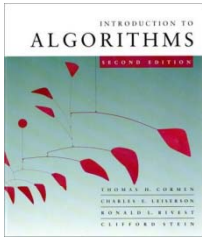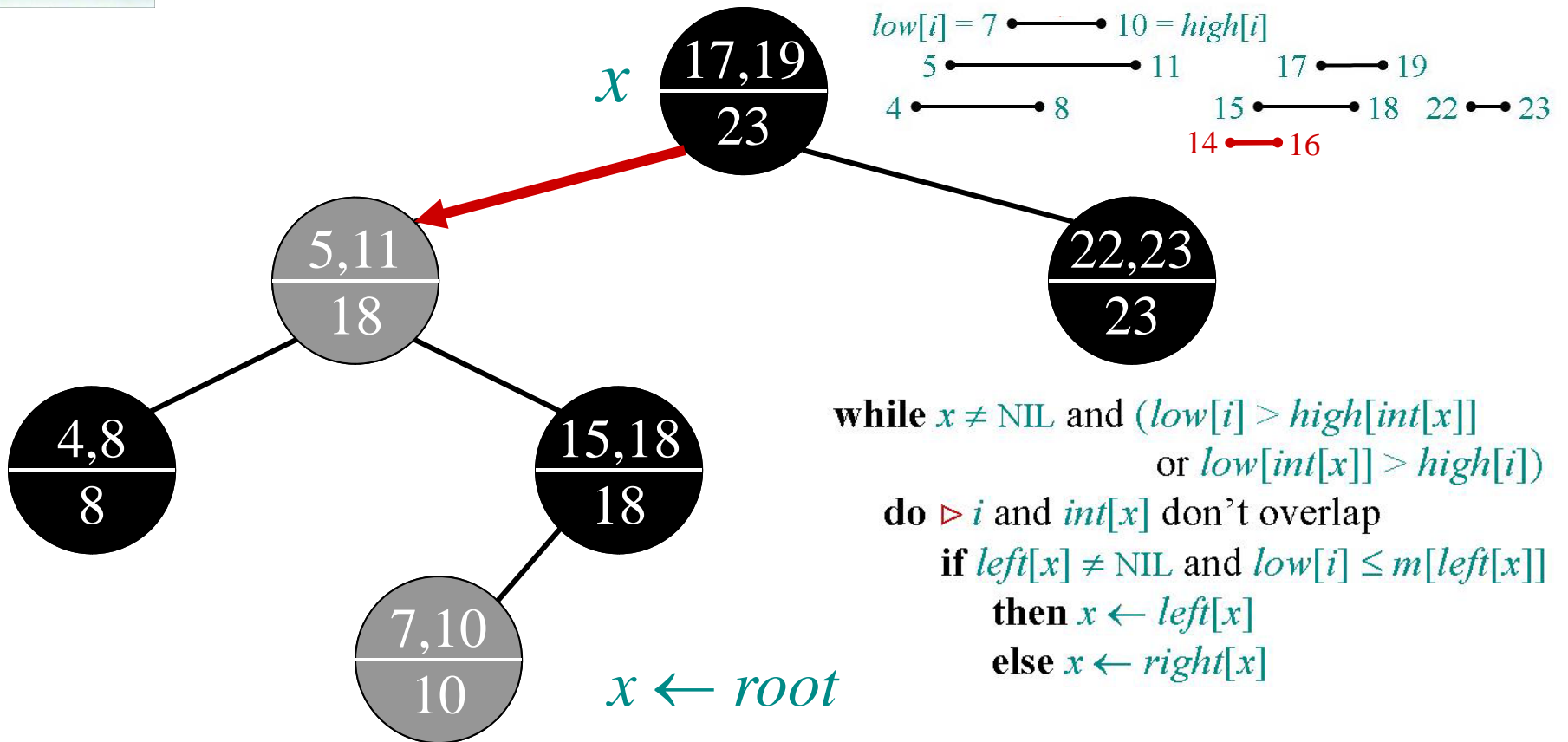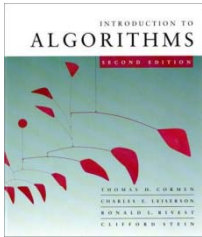   else $x \leftarrow right[x]$

$x \leftarrow root$

[14,16] and [17,19] don't overlap

$14 \leq 18 \Rightarrow x \leftarrow left[x]$

Example 1: INTERVAL-SEARCH([14,16])

$low[i] = 7 \bullet\!\!-\!\!\bullet 10 = high[i]$
$5 \bullet\!\!-\!\!\bullet 11 \qquad 17 \bullet\!\!-\!\!\bullet 19$
$4 \bullet\!\!-\!\!\bullet 8 \qquad 15 \bullet\!\!-\!\!\bullet 18 \quad 22 \bullet\!\!-\!\!\bullet 23$
$14 \bullet\!\!-\!\!\bullet 16$

17,19
—
23

x

5,11
—
18

22,23
—
23

4,8
—
8

15,18
—
18

7,10
—
10

while $x \neq$ NIL **and** $(low[i] > high[int[x]]$
$\qquad\qquad$ **or** $low[int[x]] > high[i])$
do ▷ i and $int[x]$ don't overlap
$\qquad$ **if** $left[x] \neq$ NIL **and** $low[i] \leq m[left[x]]$
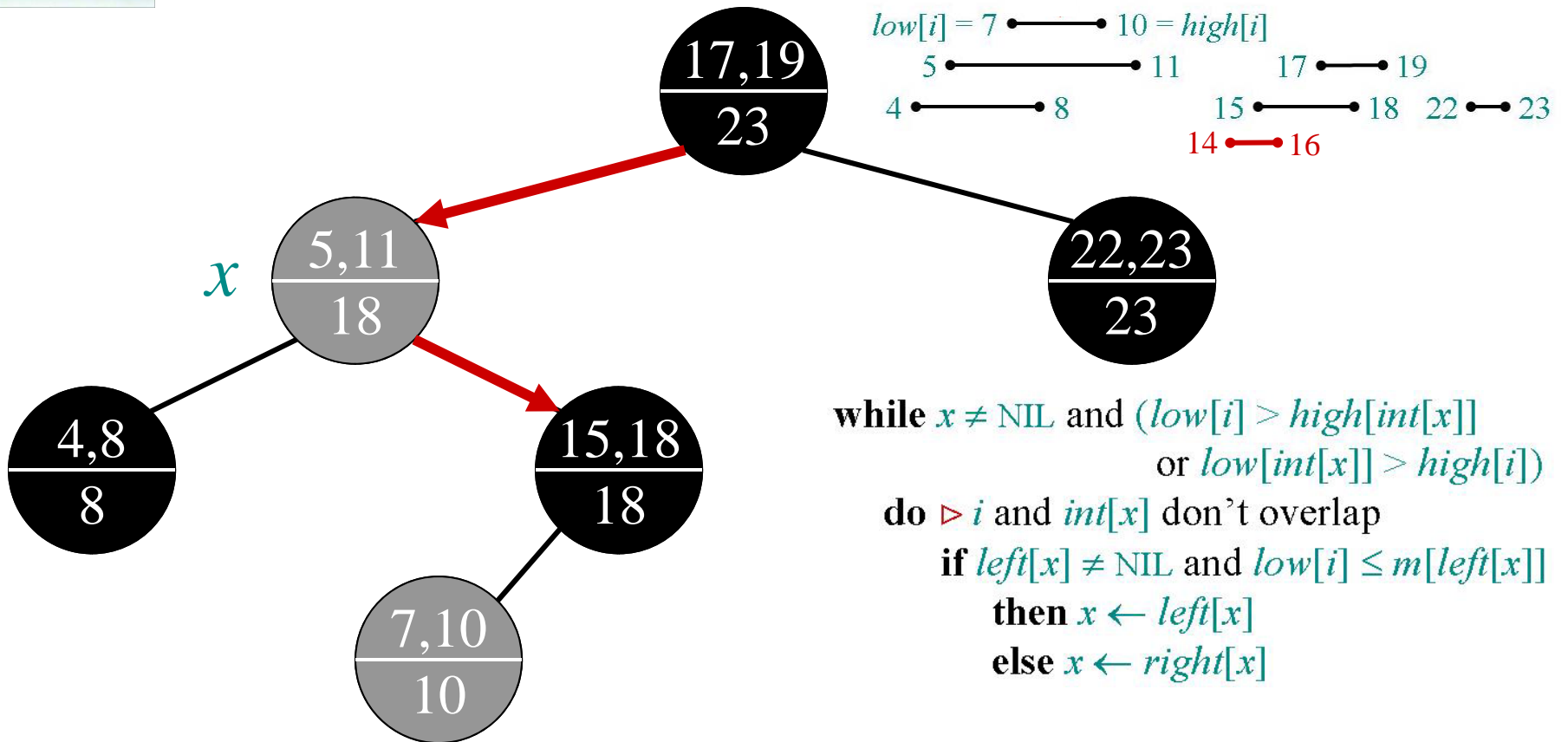$\qquad\qquad$ **then** $x \leftarrow left[x]$
$\qquad\qquad$ **else** $x \leftarrow right[x]$

[14,16] and [5,11] don't overlap

$14 > 8 \Rightarrow x \leftarrow right[x]$

Example 1: INTERVAL-SEARCH([14,16])

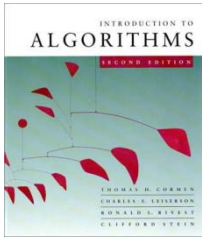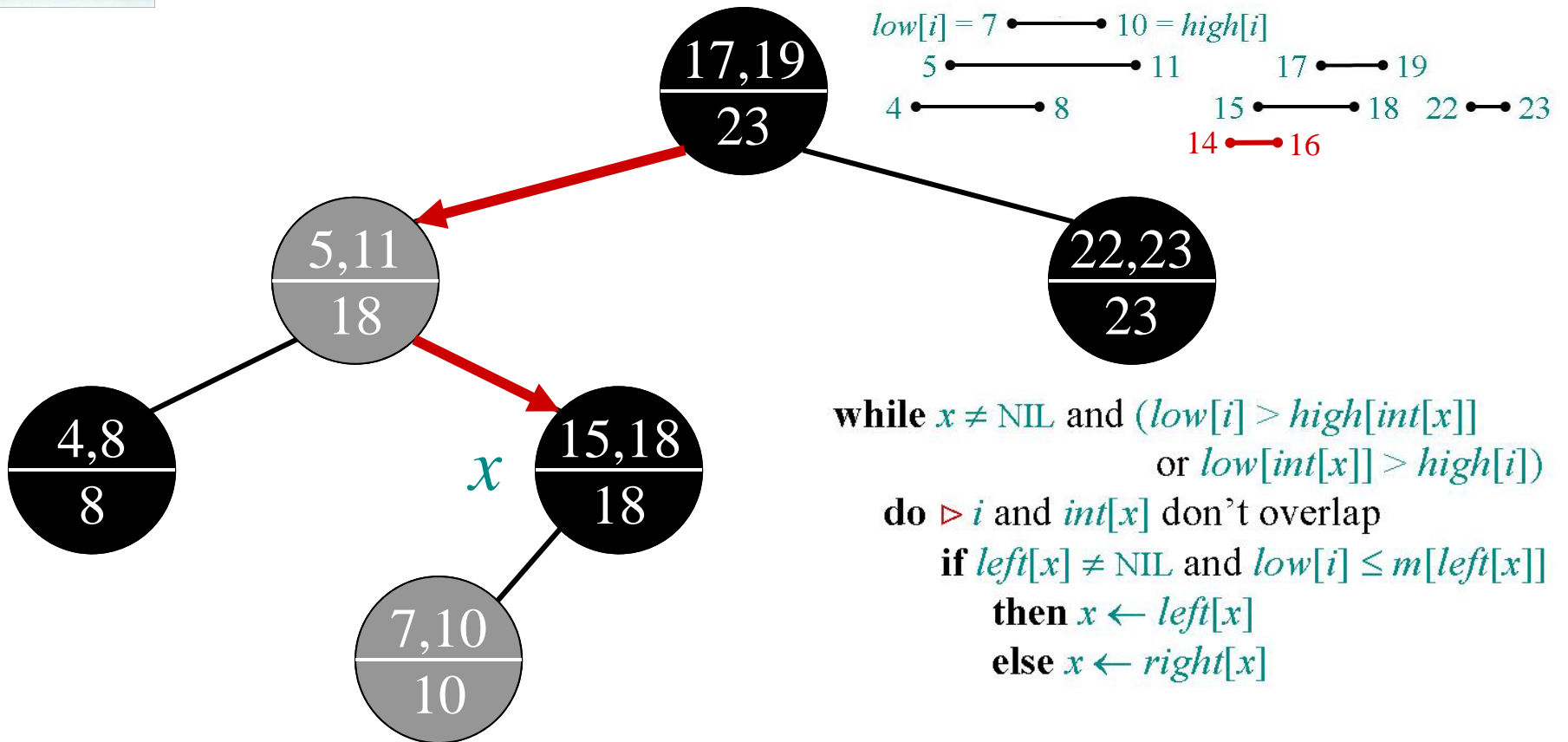$low[i] = 7 \bullet\!\!-\!\!\bullet 10 = high[i]$

$5 \bullet\!\!-\!\!-\!\!-\!\!\bullet 11 \qquad 17 \bullet\!\!-\!\!\bullet 19$

$4 \bullet\!\!-\!\!-\!\!\bullet 8 \qquad 15 \bullet\!\!-\!\!-\!\!\bullet 18 \quad 22 \bullet\!\!-\!\!\bullet 23$

$14 \bullet\!\!-\!\!\bullet 16$

while $x \neq$ NIL and $(low[i] > high[int[x]]$
$\qquad\qquad\qquad$ or $low[int[x]] > high[i])$
do $\triangleright i$ and $int[x]$ don't overlap
$\qquad$ **if** $left[x] \neq$ NIL and $low[i] \leq m[left[x]]$
$\qquad\qquad$ **then** $x \leftarrow left[x]$
$\qquad\qquad$ **else** $x \leftarrow right[x]$

[14,16] and [15,18] overlap

return [15,18]
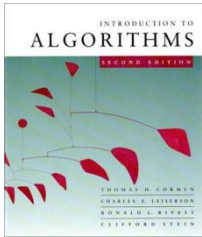
Example 2: INTERVAL-SEARCH([12,14])

$low[i] = 7$ •——————• $10 = high[i]$
5 •——————• 11 17 •——• 19
4 •——————• 8 15 •——————• 18 22 •——• 23
12 •——• 14

while $x \neq$ NIL **and** $(low[i] > high[int[x]]$
                or $low[int[x]] > high[i])$
do ▷ i and $int[x]$ don't overlap
     if $left[x] \neq$ NIL **and** $low[i] \leq m[left[x]]$
         then $x \leftarrow left[x]$
         else $x \leftarrow right[x]$

$x \leftarrow root$

[12,14] and [17,19] don't overlap

$12 \leq 18 \Rightarrow x \leftarrow left[x]$
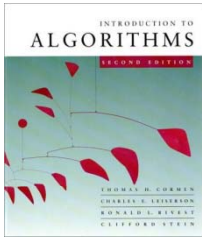
Example 2: INTERVAL-SEARCH([12,14])

$low[i] = 7 \bullet\!\!-\!\!-\!\!\bullet 10 = high[i]$

$5 \bullet\!\!-\!\!-\!\!-\!\!-\!\!\bullet 11 \qquad 17 \bullet\!\!-\!\!\bullet 19$

$4 \bullet\!\!-\!\!-\!\!-\!\!\bullet 8 \qquad\qquad 15 \bullet\!\!-\!\!-\!\!\bullet 18 \quad 22 \bullet\!\!-\!\!\bullet 23$

$12 \bullet\!\!-\!\!-\!\!\bullet 14$

17,19 / 23

x 5,11 / 18

22,23 / 23

4,8 / 8

15,18 / 18

7,10 / 10

while $x \neq$ NIL and $(low[i] > high[int[x]]$
$\qquad\qquad\qquad$ or $low[int[x]] > high[i])$
do ▷ i and $int[x]$ don't overlap
$\qquad$ **if** $left[x] \neq$ NIL and $low[i] \leq m[left[x]]$
$\qquad\qquad$ **then** $x \leftarrow left[x]$
$\qquad\qquad$ **else** $x \leftarrow right[x]$

[12,14] and [5,11] don't overlap

$12 > 8 \Rightarrow x \leftarrow right[x]$
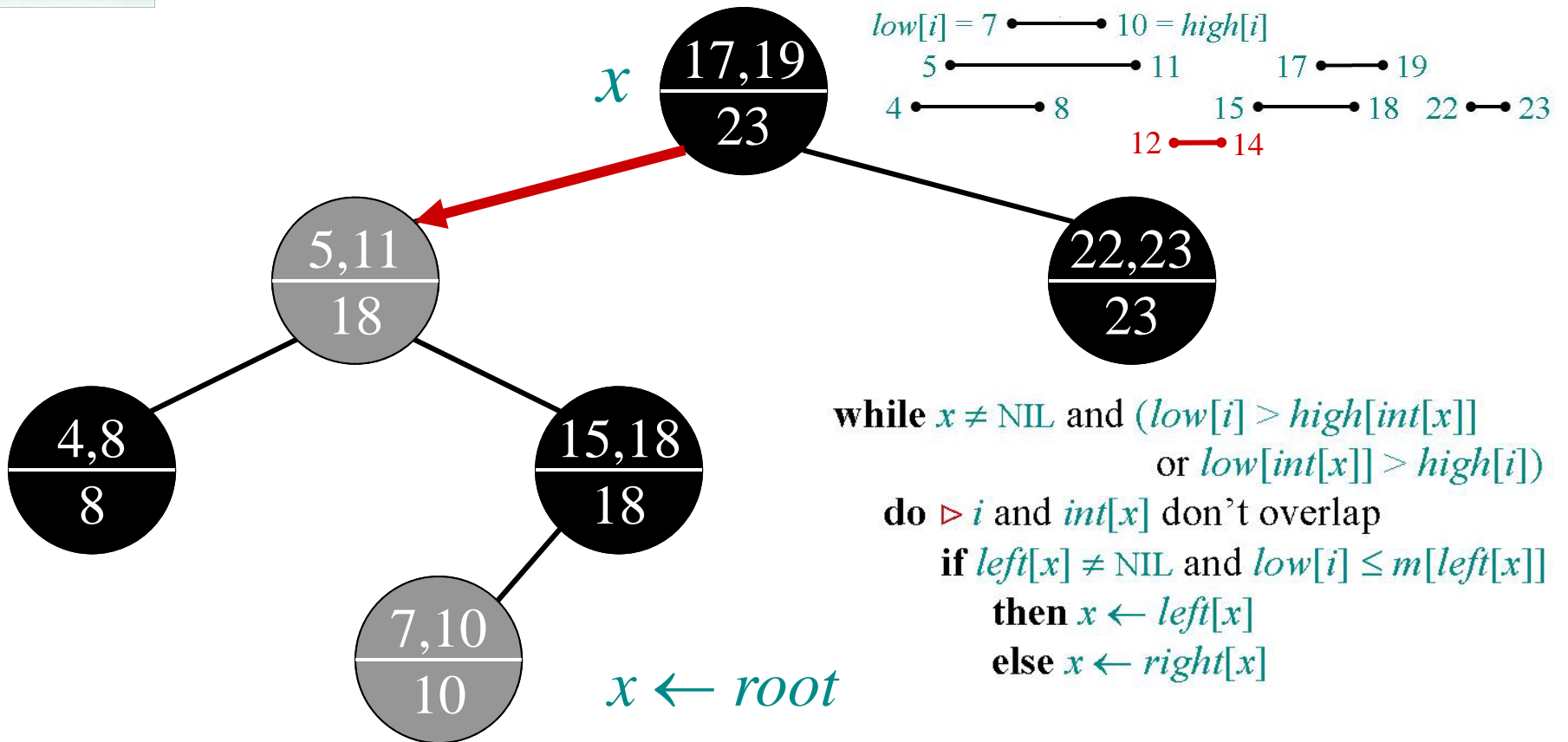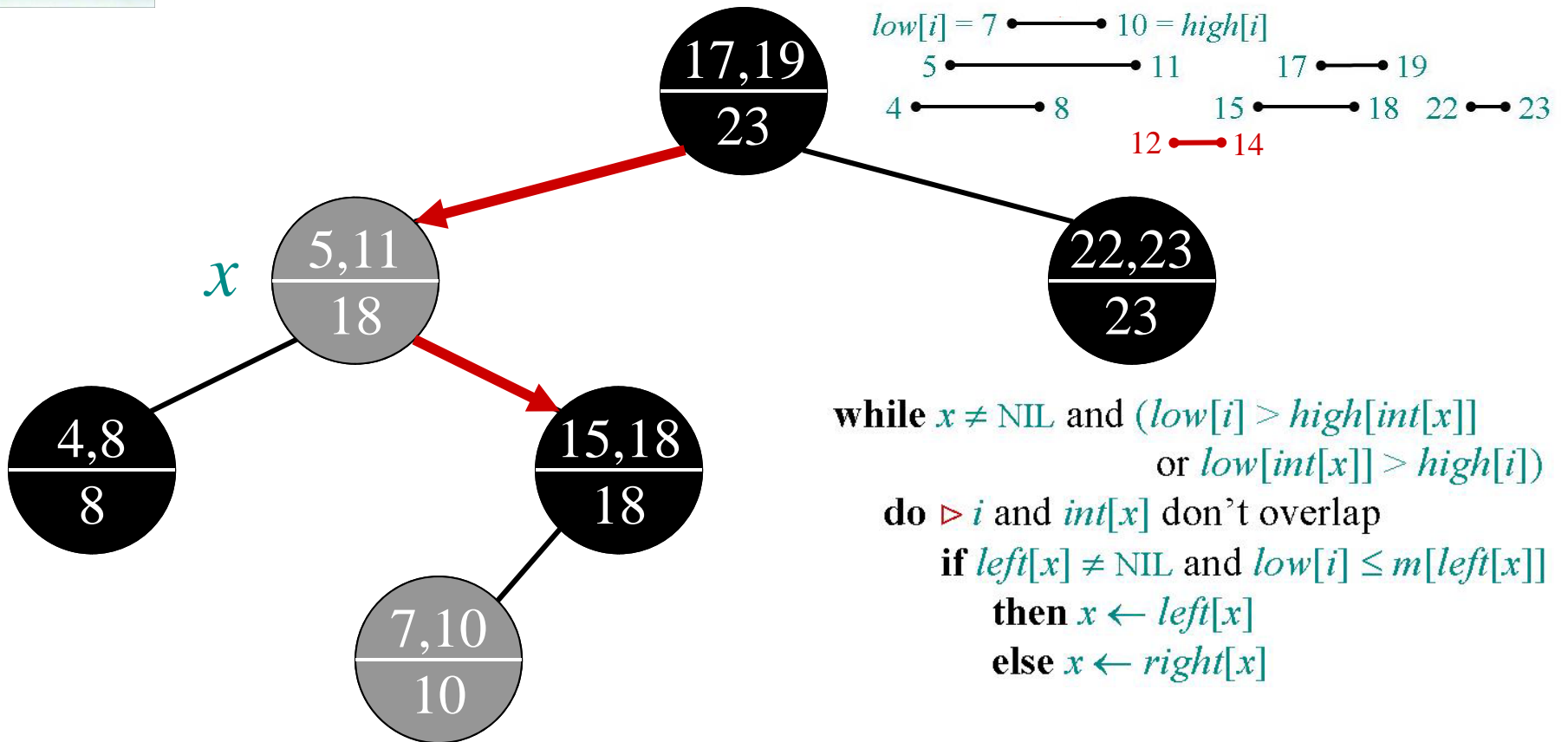
Example 2: INTERVAL-SEARCH([12,14])

$$low[i] = 7 \bullet\!\!-\!\!\bullet 10 = high[i]$$
$$5 \bullet\!\!-\!\!\bullet 11 \qquad 17 \bullet\!\!-\!\!\bullet 19$$
$$4 \bullet\!\!-\!\!\bullet 8 \qquad 15 \bullet\!\!-\!\!\bullet 18 \quad 22 \bullet\!\!-\!\!\bullet 23$$
$$12 \bullet\!\!-\!\!\bullet 14$$

17,19
23

5,11
18

22,23
23

4,8
8

15,18
x
18

7,10
10

while $x \neq$ NIL and $(low[i] > high[int[x]]$
$\qquad\qquad$ or $low[int[x]] > high[i])$
do ▷ i and $int[x]$ don't overlap
$\qquad$ **if** $left[x] \neq$ NIL and $low[i] \leq m[left[x]]$
$\qquad\qquad$ **then** $x \leftarrow left[x]$
$\qquad\qquad$ **else** $x \leftarrow right[x]$
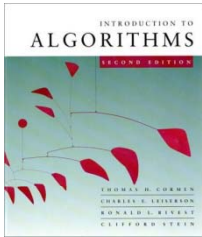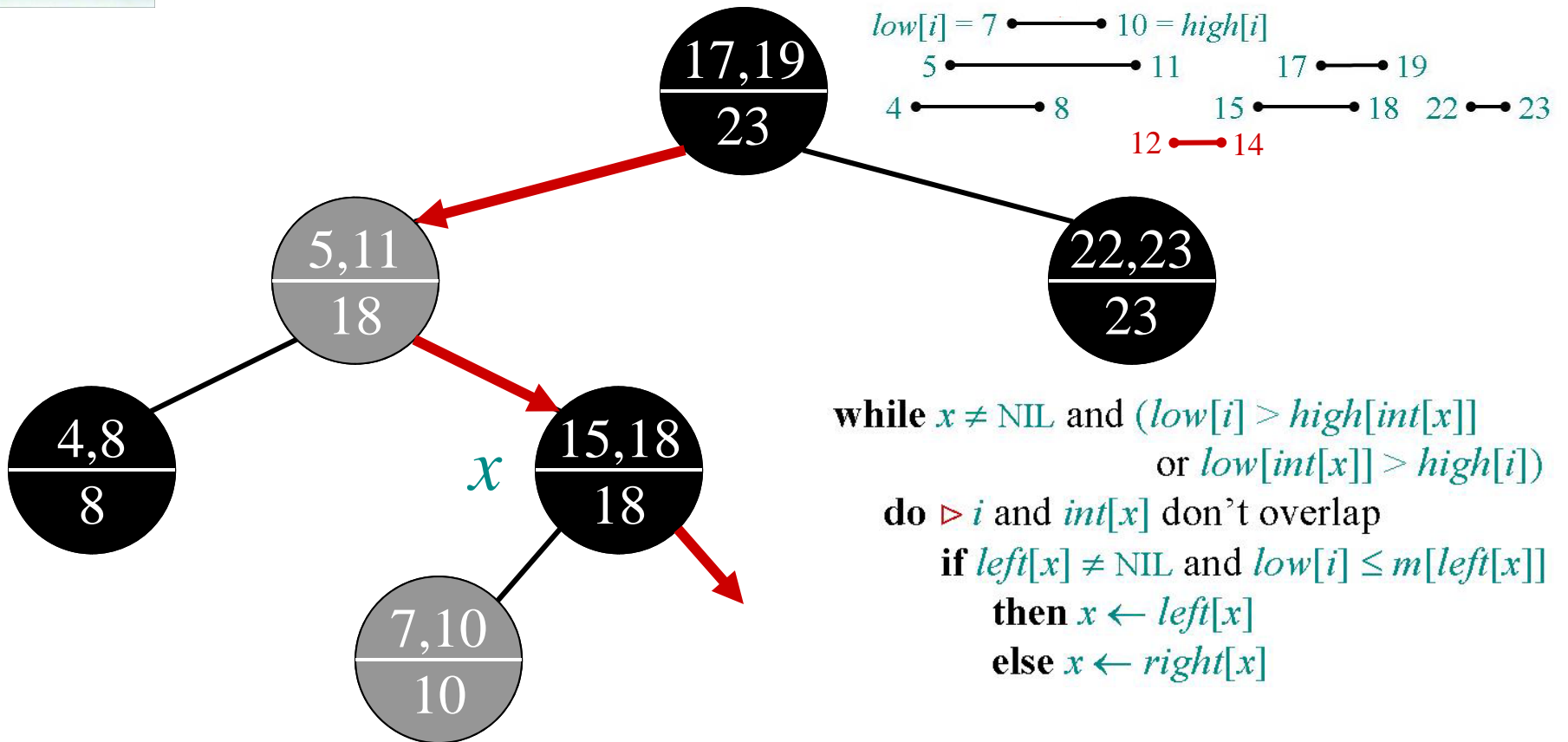
[12,14] and [15,18] don't overlap

$12 > 10 \Rightarrow x \leftarrow right[x]$
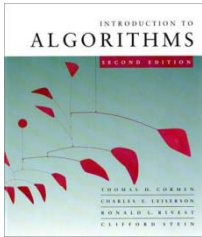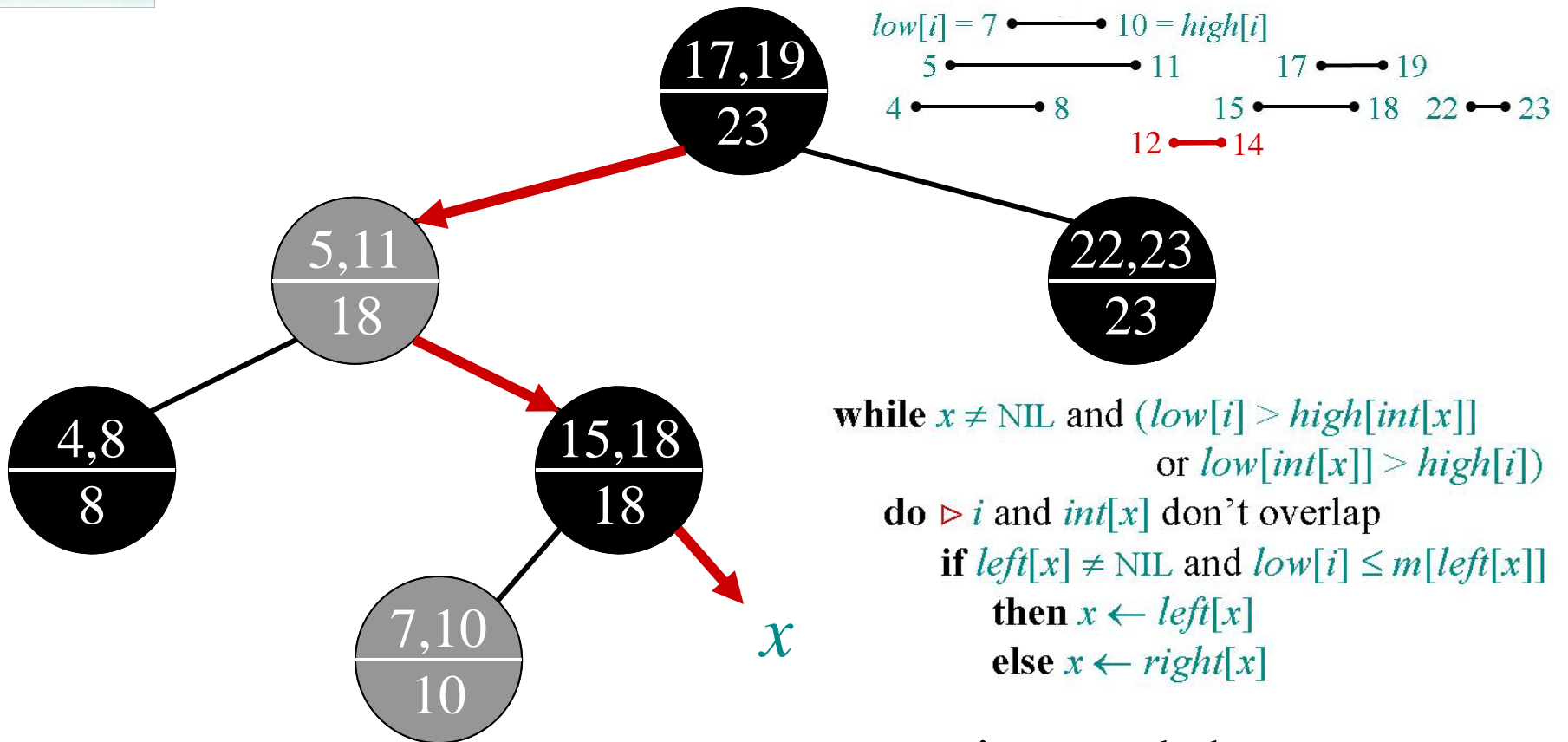
Example 2: INTERVAL-SEARCH([12,14])

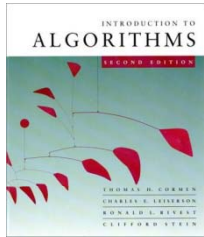$low[i] = 7 \bullet\!\!-\!\!\bullet 10 = high[i]$

$5 \bullet\!\!-\!\!-\!\!\bullet 11 \qquad 17 \bullet\!\!-\!\!\bullet 19$

$4 \bullet\!\!-\!\!\bullet 8 \qquad 15 \bullet\!\!-\!\!\bullet 18 \quad 22 \bullet\!\!-\!\!\bullet 23$

$12 \bullet\!\!-\!\!\bullet 14$

while $x \neq$ NIL **and** $(low[i] > high[int[x]]$
$\qquad\qquad\qquad$ **or** $low[int[x]] > high[i])$
do ▷ i and $int[x]$ don't overlap
$\qquad$ **if** $left[x] \neq$ NIL **and** $low[i] \leq m[left[x]]$
$\qquad\qquad$ **then** $x \leftarrow left[x]$
$\qquad\qquad$ **else** $x \leftarrow right[x]$

$x =$ NIL $\Rightarrow$ no interval that
overlaps [12,14] exists

Analysis

Time $= O(h) = O(\log n)$, since INTERVAL-SEARCH does constant work at each level as it follows a simple path down the tree.
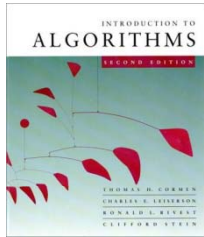
List *all* overlapping intervals:
• Search, list, delete, repeat.
• Insert them all again at the end.
Time $= O(k \log n)$, where k is the total number of overlapping intervals.

This is an ***output-sensitive*** bound.

Best algorithm to date: $O(k + \log n)$.

CS 5633 Analysis of Algorithms

Correctness

Theorem. Let L be the set of intervals in the left subtree of node x, and let R be the set of intervals in x's right subtree.
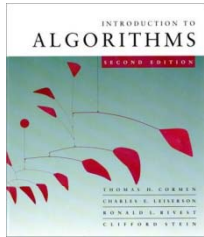
- If the search goes right, then
$$\{\, i' \in L : i' \text{ overlaps } i \,\} = \varnothing.$$

- If the search goes left, then
$$\{ i' \in L : i' \text{ overlaps } i \,\} = \varnothing$$
$$\Rightarrow \{ i' \in R : i' \text{ overlaps } i \,\} = \varnothing.$$
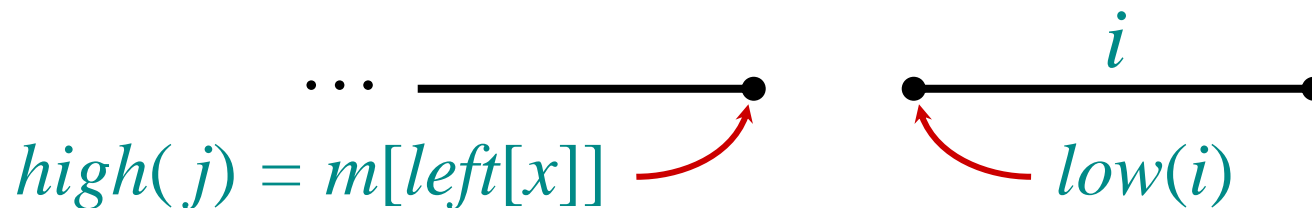
In other words, it's always safe to take only 1 of the 2 children: we'll either find something, or nothing was to be found.
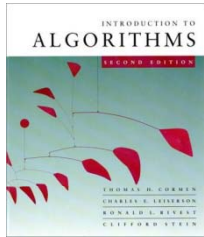
Correctness proof

Proof. Suppose first that the search goes right.

- If $left[x] = \text{NIL}$, then we're done, since $L = \varnothing$.

- Otherwise, the code dictates that we must have $low[i] > m[left[x]]$. The value $m[left[x]]$ corresponds to the right endpoint of some interval $j \in L$, and no other interval in L can have a larger right endpoint than $high(j)$.

$$\ldots \quad \bullet\!\!-\!\!-\!\!-\!\!-\!\!-\!\!\bullet \qquad \overset{i}{\bullet\!\!-\!\!-\!\!-\!\!-\!\!-\!\!\bullet}$$

$$high(j) = m[left[x]] \qquad\qquad low(i)$$

- Therefore, $\{i' \in L : i' \text{ overlaps } i\} = \varnothing$.

Proof (continued)

Suppose that the search goes left, and assume that
$$\{ i' \in L : i' \text{ overlaps } i \} = \varnothing.$$

- Then, the code dictates that $low[i] \le m[left[x]] = high[j]$ for some $j \in L.$
- Since $j \in L$, it does not overlap i, and hence $high[i] < low[j].$
- But, the binary-search-tree property implies that for all $i' \in R$, we have $low[j] \le low[i'].$
- But then $\{ i' \in R : i' \text{ overlaps } i \} = \varnothing.$ ▨

CS 5633 Analysis of Algorithms