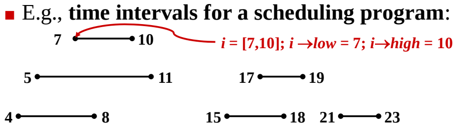
Interval and Segment Trees

- **Segment tree** stores intervals, and optimized for "***which of these intervals contains a given point*" queries.**
- **Interval tree** stores intervals as well, but optimized for "***which of these intervals overlap with a given interval***" queries.
- **Range tree** stores points, and optimized for "*which points fall within a given interval*" queries.
- **Binary indexed tree** stores items-count per index, and optimized for "***how many items are there between index m and n***" queries.

Interval Trees

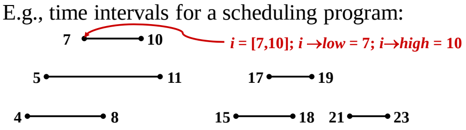
- The problem: maintain a set of intervals
  - E.g., **time intervals for a scheduling program**:



***i* = [7,10]; *i* →*low* = 7; *i*→*high* = 10**

Interval Trees

- The problem: maintain a set of intervals
  - E.g., time intervals for a scheduling program:



***i* = [7,10]; *i* →*low* = 7; *i*→*high* = 10**

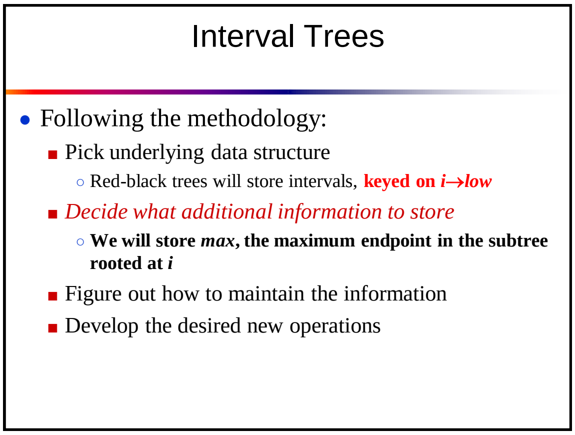
  - Query: find an interval in the set that overlaps a given query interval
    - [14,16] → [15,18]
    - [16,19] → [15,18] or [17,19]
    - [12,14] → NULL

Interval Trees

- **Methodology for Augmenting Data Structures**
  - Pick underlying data structure
  - Decide what additional information to store
  - Figure out how to maintain the information
  - Develop the desired new operations

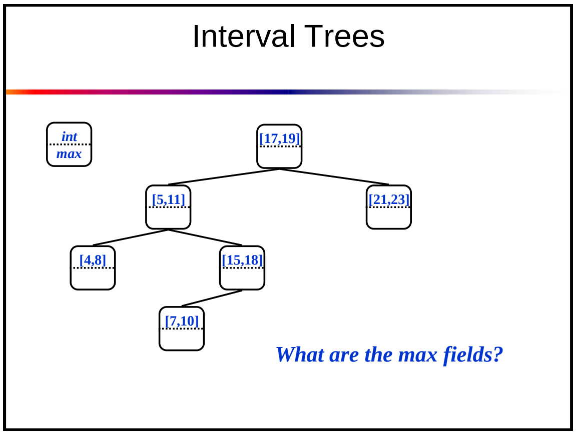
Interval Trees

- Following the methodology:
  - *Pick underlying data structure*
    - **Red-black trees will store intervals, keyed on *i*→*low***
  - Decide what additional information to store
  - Figure out how to maintain the information
  - Develop the desired new operations



Interval Trees

- Following the methodology:
  - Pick underlying data structure
    - Red-black trees will store intervals, **keyed on *i*→*low***
  - *Decide what additional information to store*
    - **We will store *max*, the maximum endpoint in the subtree rooted at *i***
  - Figure out how to maintain the information
  - Develop the desired new operations



Interval Trees

int
max

[17,19]

[5,11]

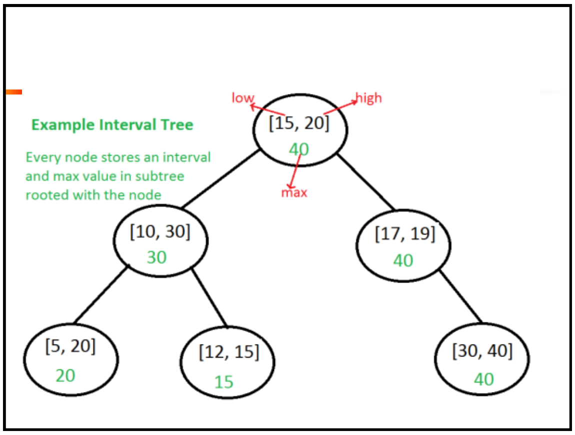
[21,23]

[4,8]

[15,18]

[7,10]

What are the max fields?



Example Interval Tree

Every node stores an interval and max value in subtree rooted with the node

low

high

max

[15, 20]
40

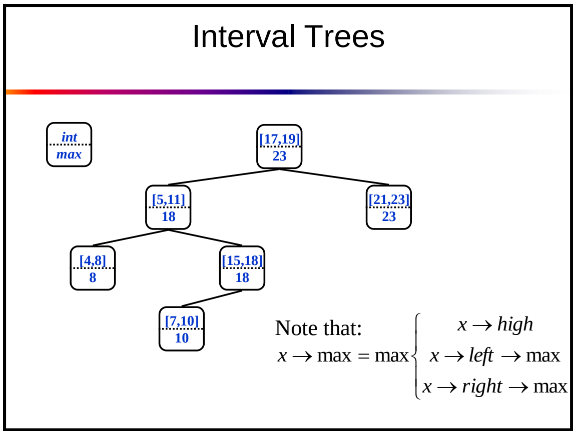
[10, 30]
30

[17, 19]
40

[5, 20]
20

[12, 15]
15

[30, 40]
40



Interval Trees

int
max

[17,19]
23

[5,11]
18

[21,23]
23

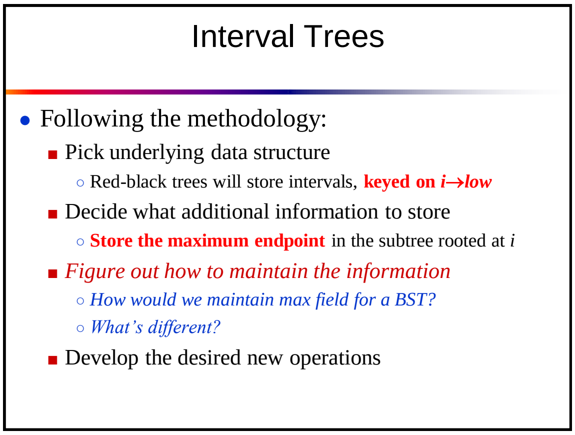
[4,8]
8

[15,18]
18

[7,10]
10

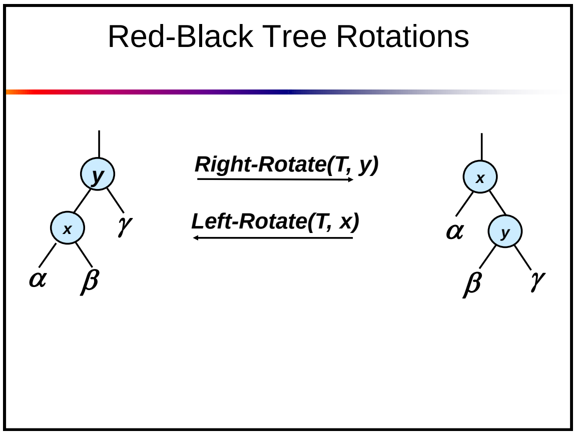
Note that:

$$x \to \max = \max\begin{cases} x \to high \\ x \to left \to \max \\ x \to right \to \max \end{cases}$$



Interval Trees

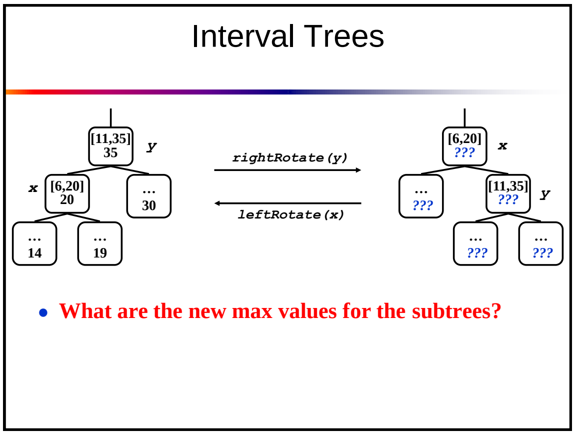
- Following the methodology:
  - Pick underlying data structure
    - Red-black trees will store intervals, **keyed on *i*→*low***
  - Decide what additional information to store
    - **Store the maximum endpoint** in the subtree rooted at *i*
  - *Figure out how to maintain the information*
    - *How would we maintain max field for a BST?*
    - *What's different?*
  - Develop the desired new operations



Red-Black Tree Rotations

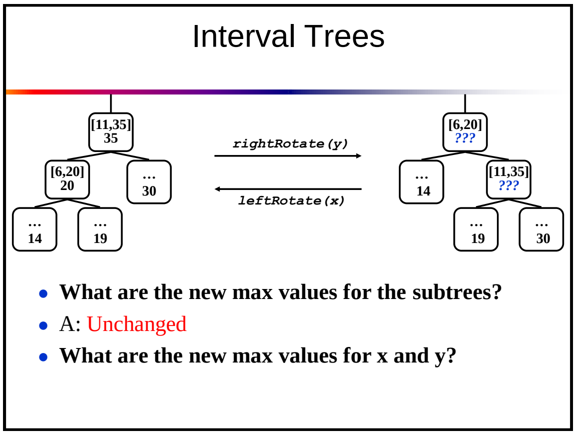
Right-Rotate(T, y)

Left-Rotate(T, x)



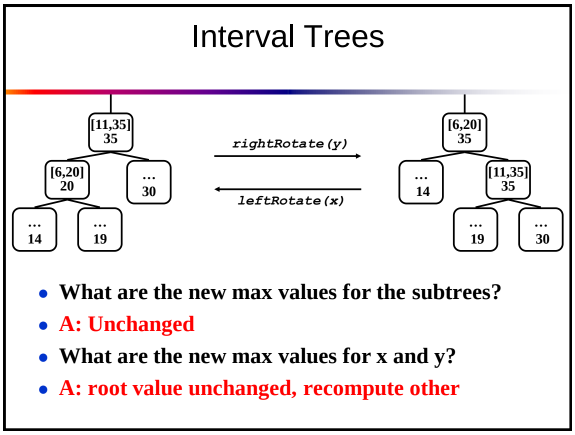
Interval Trees

- **What are the new max values for the subtrees?**



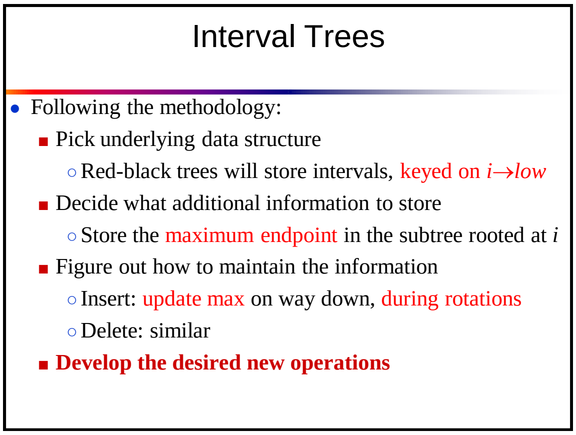
Interval Trees

- **What are the new max values for the subtrees?**
- A: Unchanged
- **What are the new max values for x and y?**



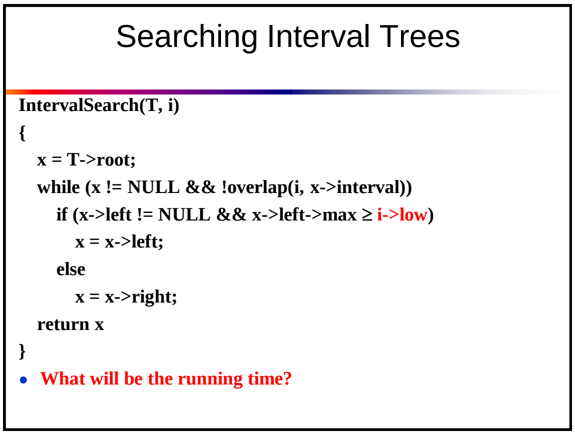
Interval Trees

- **What are the new max values for the subtrees?**
- **A: Unchanged**
- **What are the new max values for x and y?**
- **A: root value unchanged, recompute other**



Interval Trees

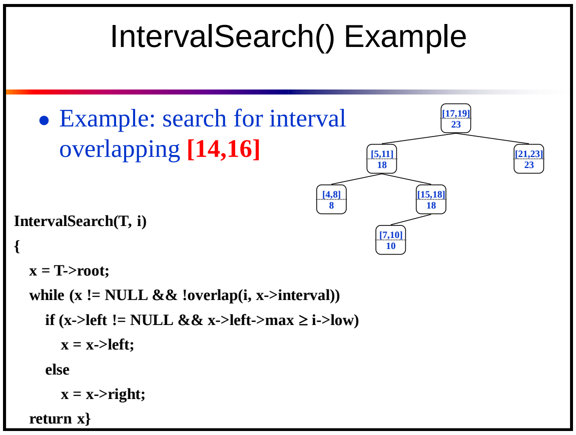
- Following the methodology:
  - Pick underlying data structure
    - Red-black trees will store intervals, keyed on *i*→*low*
  - Decide what additional information to store
    - Store the maximum endpoint in the subtree rooted at *i*
  - Figure out how to maintain the information
    - Insert: update max on way down, during rotations
    - Delete: similar
  - **Develop the desired new operations**



Searching Interval Trees

```
IntervalSearch(T, i)
{
    x = T->root;
    while (x != NULL && !overlap(i, x->interval))
        if (x->left != NULL && x->left->max ≥ i->low)
            x = x->left;
        else
            x = x->right;
    return x
}
```

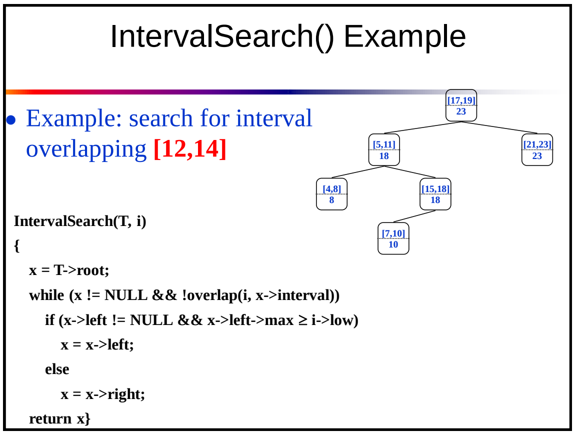
- **What will be the running time?**



IntervalSearch() Example

- Example: search for interval overlapping **[14,16]**

```
IntervalSearch(T, i)
{
  x = T->root;
  while (x != NULL && !overlap(i, x->interval))
    if (x->left != NULL && x->left->max ≥ i->low)
      x = x->left;
    else
      x = x->right;
  return x}
```



IntervalSearch() Example

- Example: search for interval overlapping **[12,14]**

```
IntervalSearch(T, i)
{
   x = T->root;
   while (x != NULL && !overlap(i, x->interval))
      if (x->left != NULL && x->left->max ≥ i->low)
         x = x->left;
      else
         x = x->right;
   return x}
```