Trees and BSTs

Interval BSTs

Paolo Camurati and Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

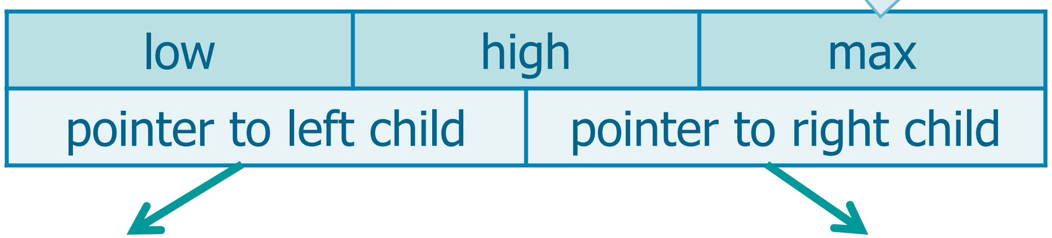
Interval BSTs

- Interval BSTs are BSTs used to store close intervals
- A close interval is
  - An ordered real couple $[t_1, t_2]$, where
    - $t_1 \leq t_2$ and
    - $[t_1, t_2] = \{t \in \Re : t_1 \leq t \leq t_2\}$
  - Open and half-open intervals omit both or one of the endpoints from the set
  - Extending our result to those intervals would be straighforward

Interval BSTs

- The interval item [t_1, t_2] can be realized with a **struct** with fields
  - low = t_1, high = t_2, and max

The maximum **high** value in the tree rooted at that node




```
typedef struct node *link;
struct node {
  float low, high, max;
  link l;
  link r;
};
```

ADT

Interval BSTs

- Intervals i and i’ have intersection iff
  - low[i] ≤ high[i’] AND low[i’] ≤ high[i]
- ∀ i, i’ the following conditions stand
  - i and i’ have an intersection

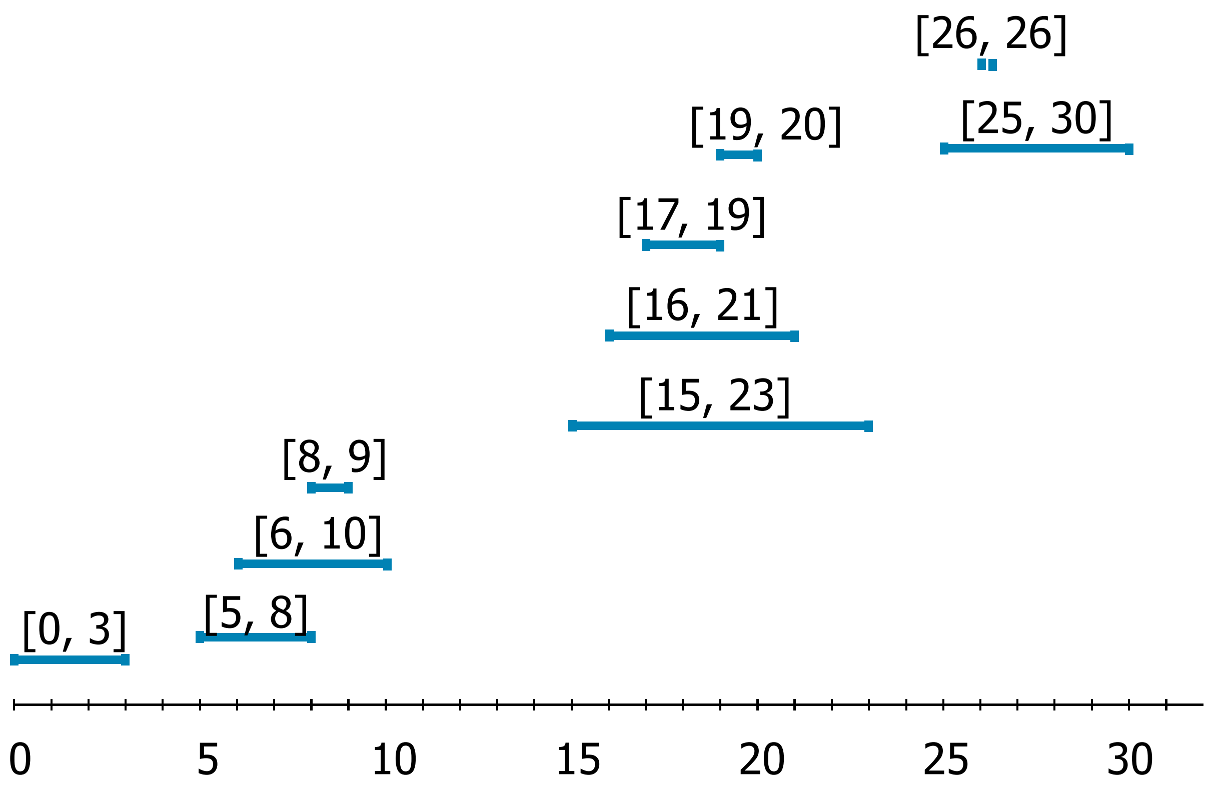
  - low[i] ≤ high[i’] AND low[i’] > high[i]

  - low[i’] ≤ high[i] AND low[i] > high[i’]

Interval tricotomy

Example

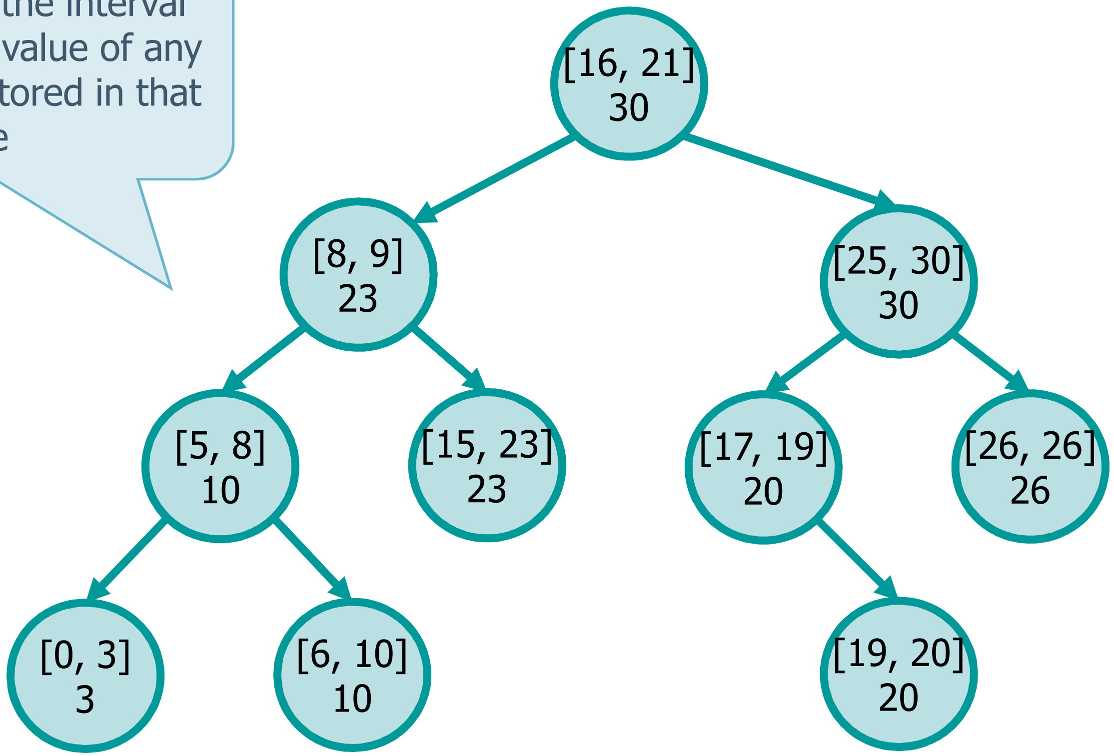
- A set of interval sorted by left enpoint



Example

- The same set of intervals into an Interval-BST

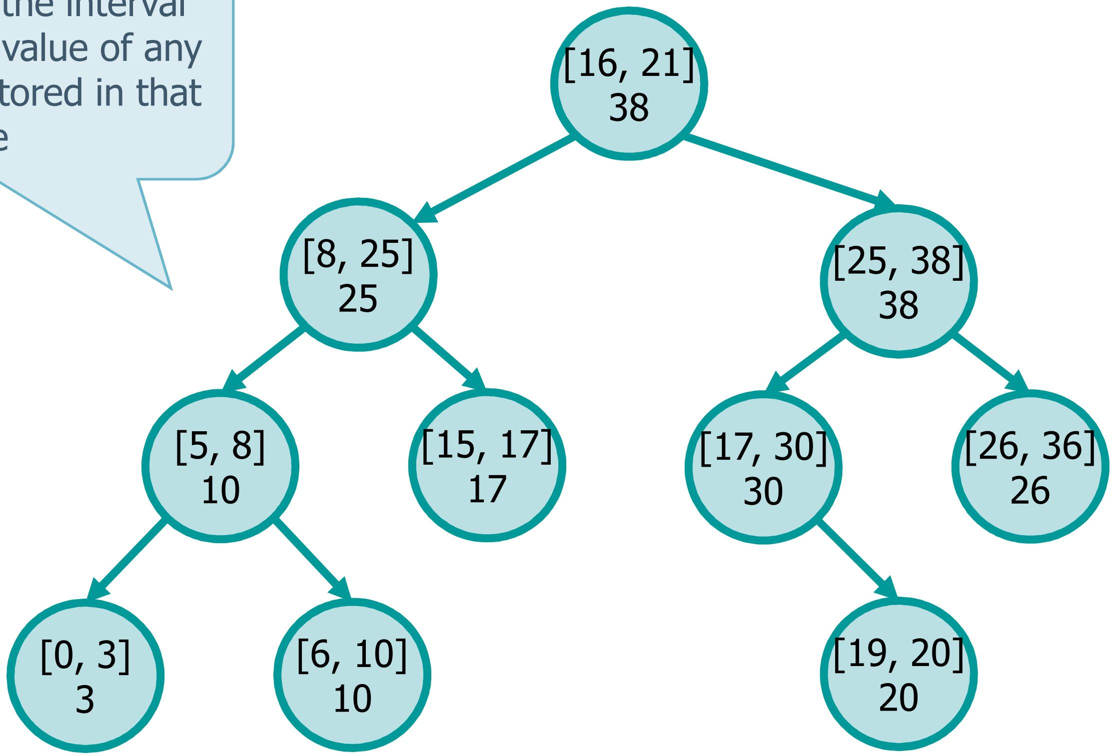
Each node stores the interval and the maximum value of any interval endpoint stored in that subtree



Example

- Another Interval-BST

Each node stores the interval and the maximum value of any interval endpoint stored in that subtree



Operations

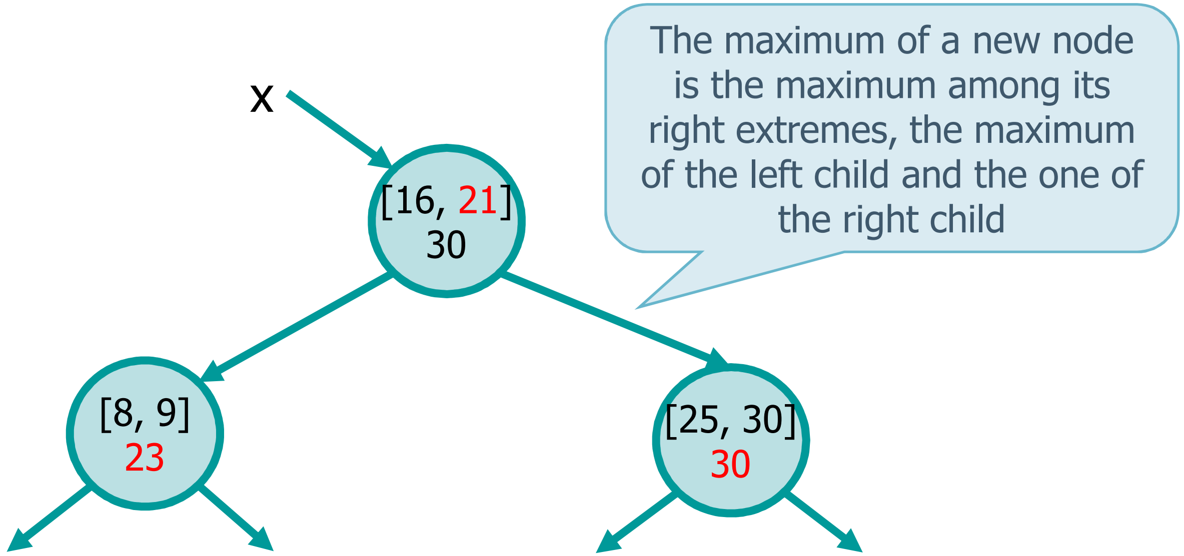
- As with BSTs with Interval-BSTs the following operations are possible
  - Insert an item (interval) into the Interval BST
    - void IBST_insert (IBST, Item) ;
  - Delete an item (interval) from the Interval BST
    - void IBST_delete (IBST, Item) ;
  - Search an item (interval) into the Interval BST and return the **first** interval with an intersection
    - Item IBST_search (IBST, Item) ;

Insert

- To insert a new node into an I-BST
  - It is sufficient to use a "standard" BST insertion procedure "working" on the **left endpoint**
  - It is necessary to determine the **maximum value** for each new node
- An inorder tree walk of the tree lists the nodes in sorted order by left endpoint

Insert: Evaluation of the maximum

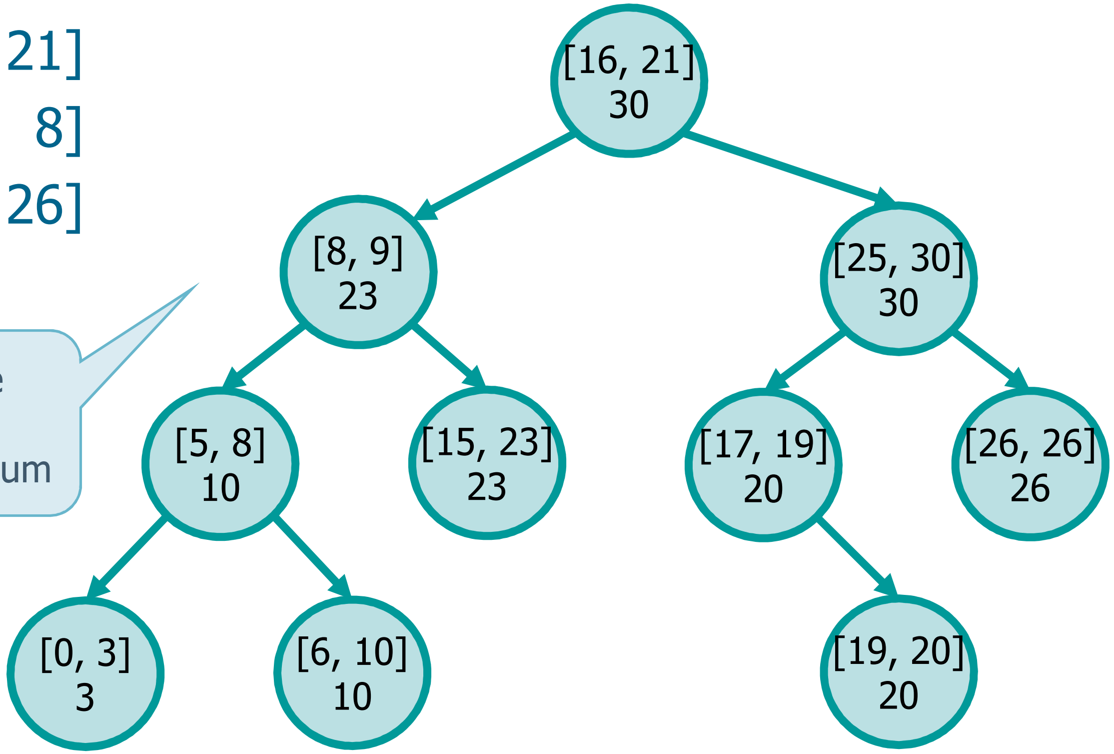
- The evaluation of the maximum has complexity Θ(1) for each new node inserted
  - x->max = max (high(x), x->left->max, x->right->max)



Examples

- Given the following Interval-BST insert nodes with intervals
  - [12, 21]
  - [4, 8]
  - [24, 26]

Insert the node
AND
update the maximum



Delete

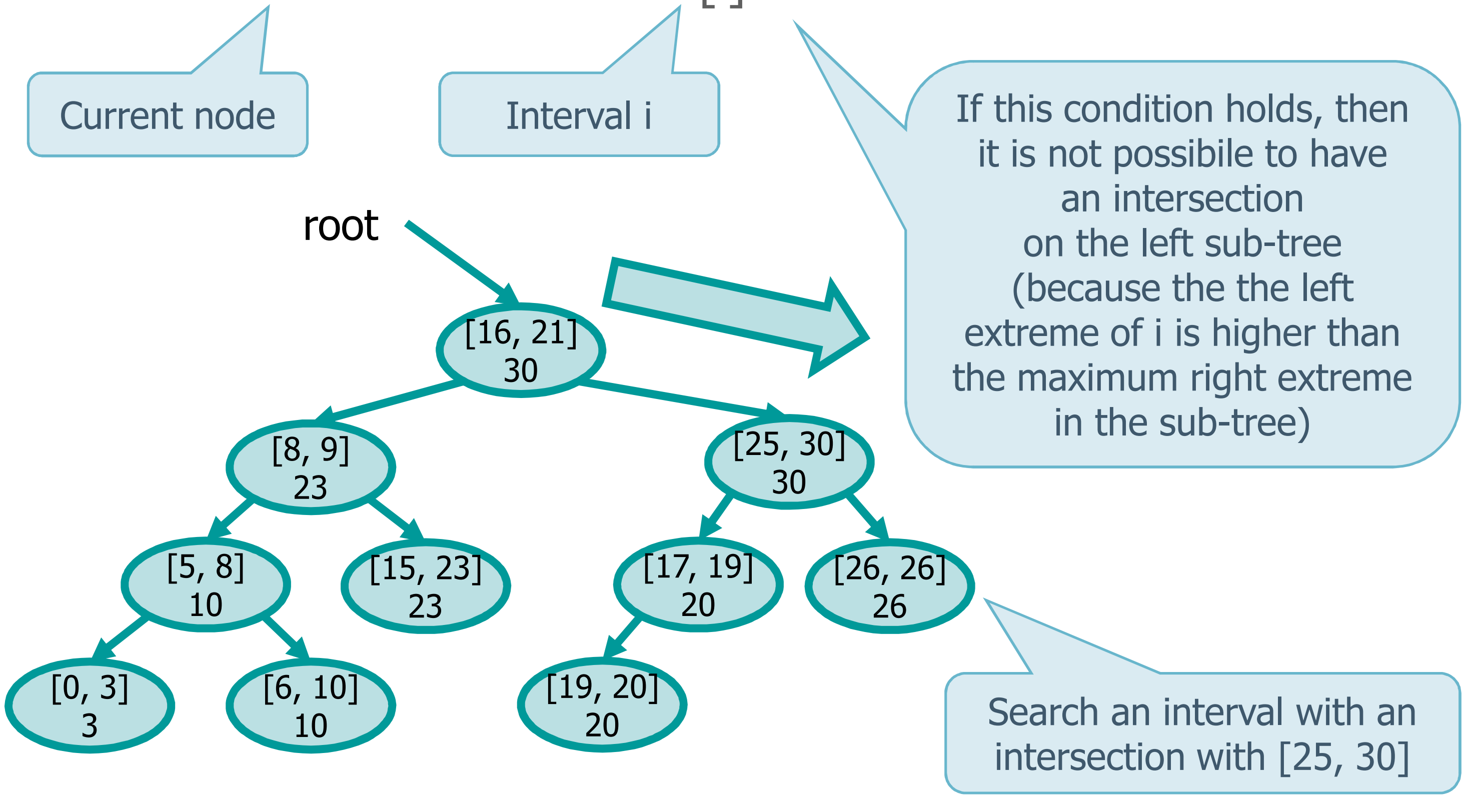
- To delete a node within an Interval-BST it is necessary to do two steps
  - Search the element to delete and
  - Delete it
- Seach is the only new operation we have to develop
- Delete, once the element has been found, can be performed using the "standard" approach presented with BSts

Search

- On an Interval-BST, when we search for an interval i usually we look-for a node n with an interval having an intersection with interval i
- The algorithm works as follow
  - Visit the tree from root
  - Termination
    - Find an interval with an intersection with i or
    - An empty tree has been reached
  - Recursion from node n
    - On the right sub-tree
    - On the left sub-tree

Search

- We recur on the right sub-tree if
  - n->l==NULL OR low[i] > n->l->max



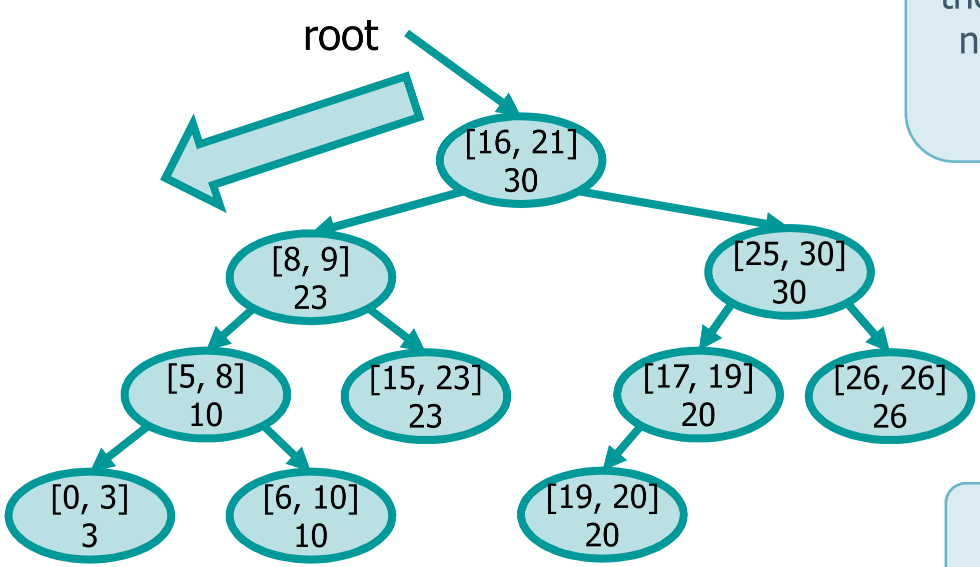
Search

- We recur on the left sub-tree if
  - n->l!=NULL AND low[i] ≤ n->l->max

Interval i

Current node

If this condition holds, then if there is no intersection on the left sub-tree then there is no intersection on the right sub-tree as well.
Why is that?



Search an interval with an intersection with [21, 27]

Search

- We recur on the left sub-tree if
  - low[i] ≤ n->l->max

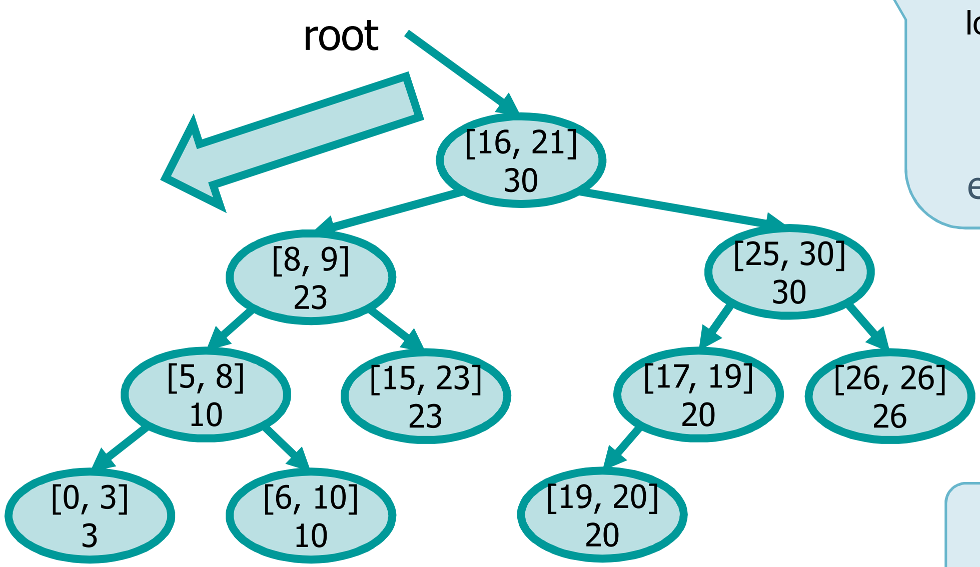
Interval i

Current node

There is no intersection on the left when

low[i] high[i]

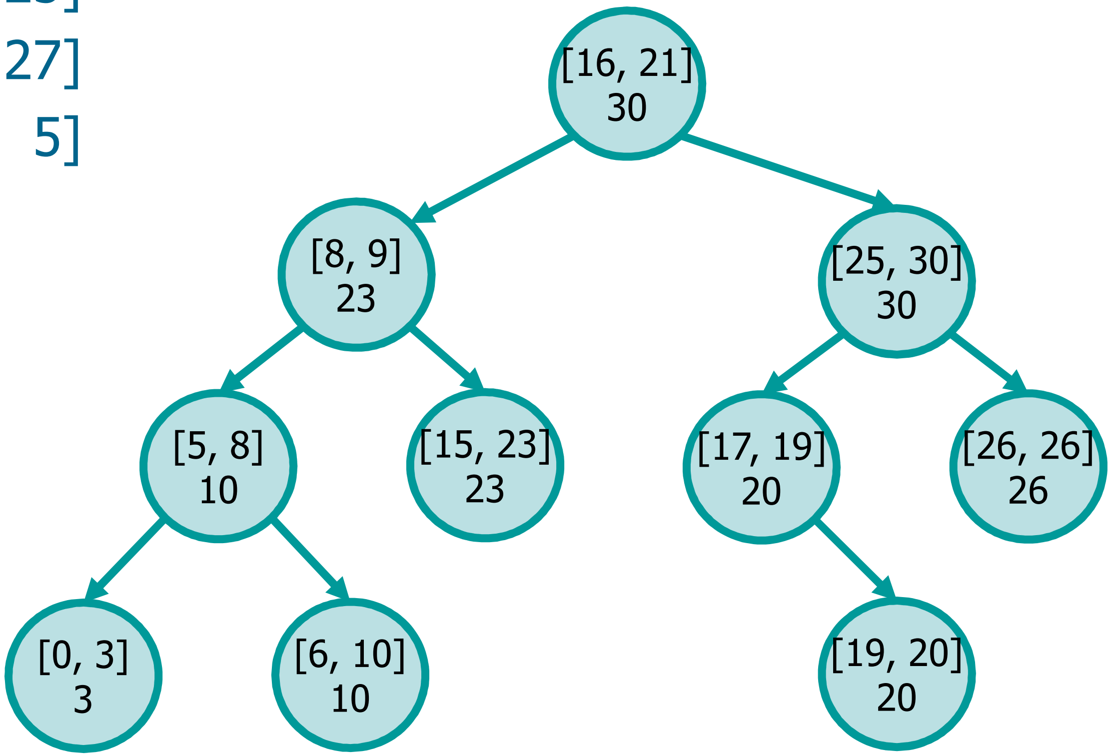
Then, on the right low endpoints are even higher



Search an interval with an intersection with [21, 27]

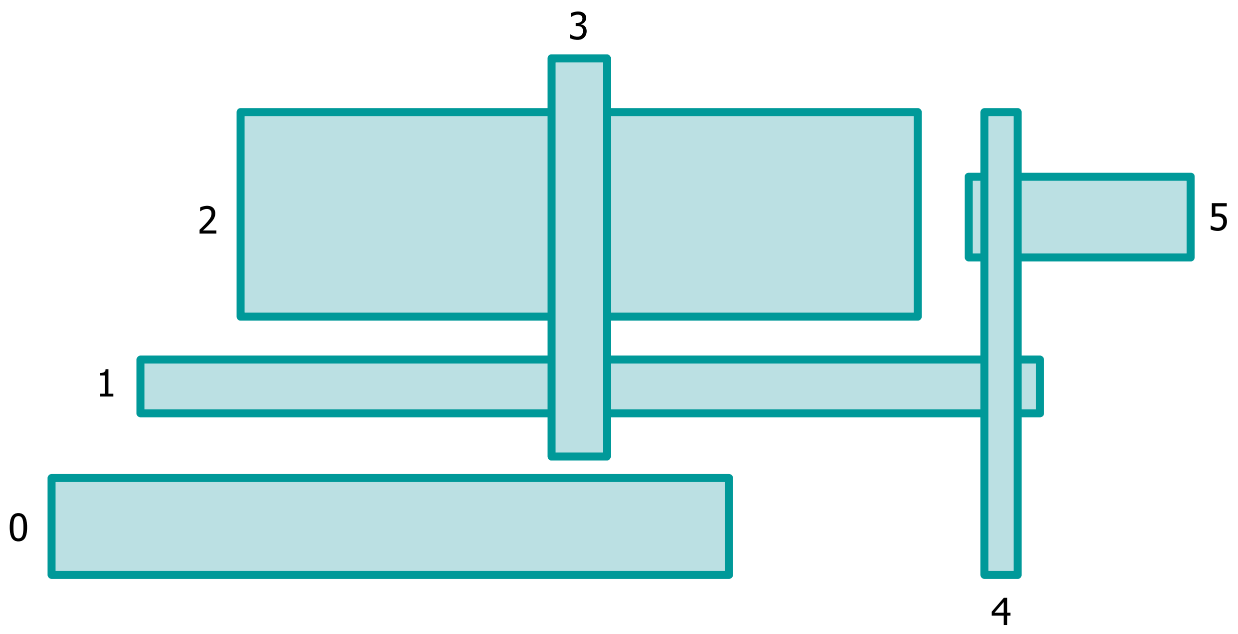
Examples

- Given the following Interval-BST search intervals
  - [22, 25]
  - [24, 27]
  - [4, 5]



Application

- Given N rectangles placed with sides parallel to the Cartesian axis
  - Check whether a new rectangle has an intersection with another rectangle



Application

- Electronic CAD Application
  - Verify if the there are connections with an intersection on an electronic circuit
- Basic Algorithm
  - Check the intersection among all rectangle couples
  - Complexity $O(N^2)$

Application

- Algorithms using IBST
  - Order rectangles based on ascending left extreme x-values
  - Iterate on rectangles for ascending x-values
    - When a left extreme in encountered, **insert** the y-value range into an I-BST and check for intersactions
    - When a right extreme is found, **remove** the interval from the I-BST y-values
  - Efficient algorithm
    - Complexity O(N·logN)
    - Applicability to VLSI and beyond

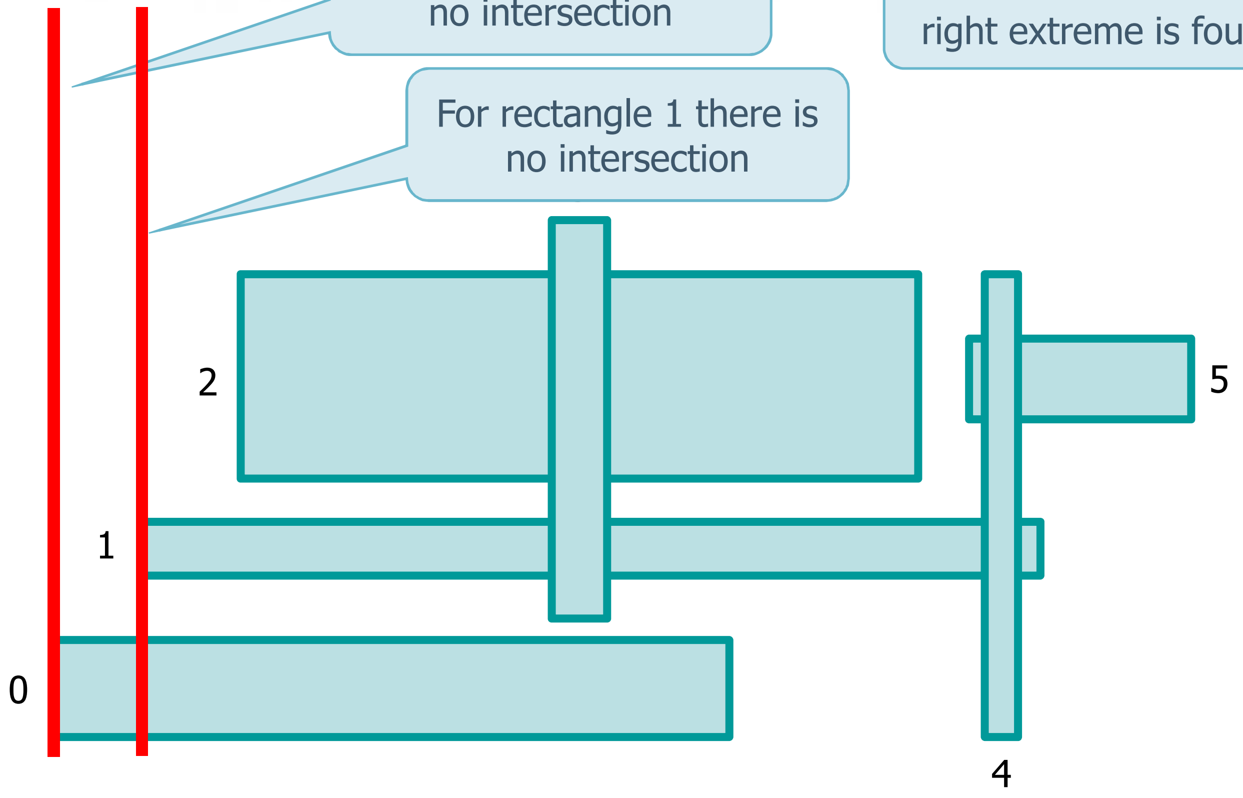
Application

Insert interval on y-values
Start from rectangle 0 on

For rectangle 0 there is no intersection

Remove intervals when right extreme is found

For rectangle 1 there is no intersection



Complexity Analysis

- Sorting
  - O(N·log N)
- If the IBST is balanced
  - Each insertion/deletion of an interval or seach of the first interval that intersects a given one has cost O(log N)
  - Searching for all intervals that intersect a given one has cost O(R logN), where R is the number of intersections

Exercise

- Given an initially empty Interval-BST, perform the following insertions
  - [3,10] [4,6] [1,3] [16,21] [7,11] [2,8] [12,19] [5,15] [9,10]
- Then, search intervals
  - [11,13]
  - [23,25]
  - [18,21]

Exercise

- Given an initially empty Interval-BST, perform the following insertions
  - [3,5] [7,9] [1,2] [2,4] [8,12] [13,21] [14,17] [11,15] [0,1]
- Then, search intervals
  - [20,25]
  - [16,19]