COL 106 Lecture 16

Topic : Binary Search Trees:

Traversal , Search, Insertion

Data Structures for Sets.

U : universal set; possibly infinite.

S a finite subset of U

Unordered dictionaries

Operations :

- ☑ List
- ☑ Search
- ☑ Add element (Insertion)
- ☐ Remove element (Deletion)

Ordered dictionaries:
Assumption: '$\leq$' is an order on U.

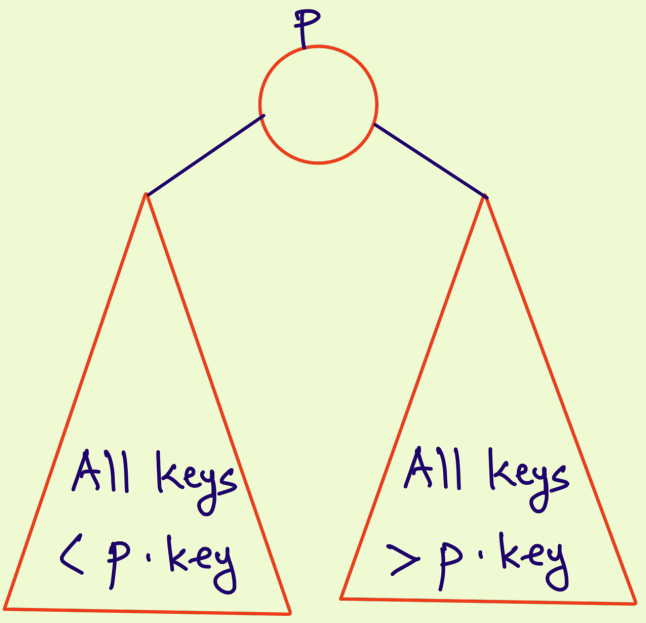
Operations: All unordered dictionary operations +

- ☑ List in sorted order
- ☐ Min, Max, predecessor, successor

Binary Search Trees

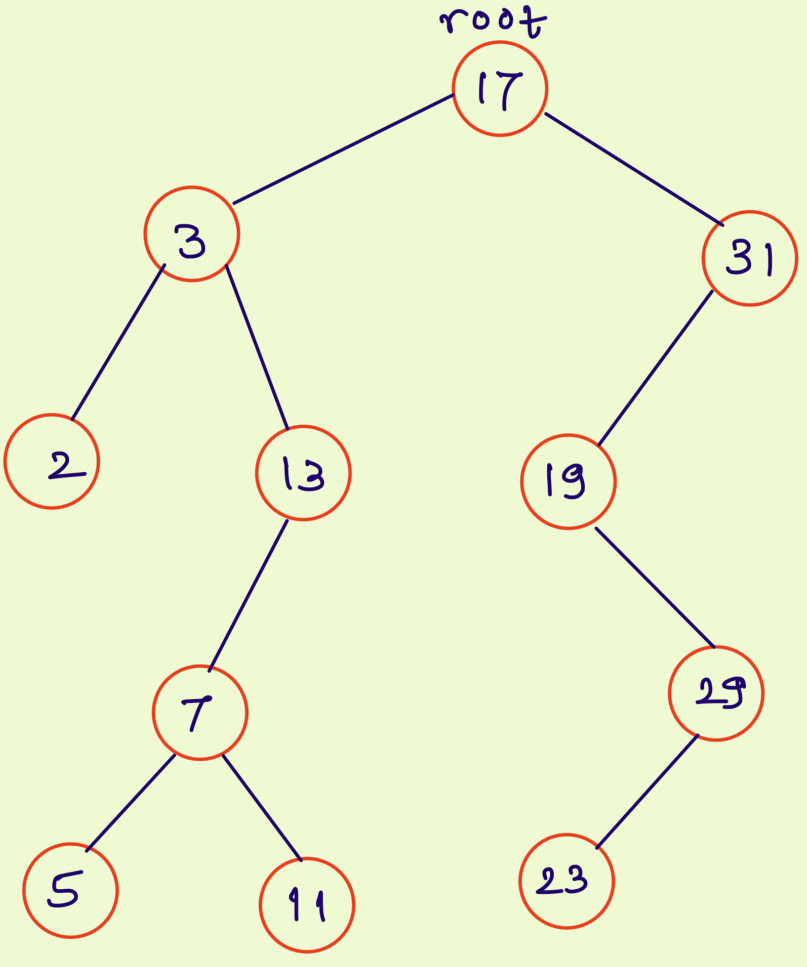
Definition: Let $(U, <)$ be an ordered set. A binary tree with keys from U is said to be a binary search tree if for every node p the following are true.

1. For every node q in the left subtree of p:
   $q.key < p.key$
2. For every node in the right subtree of p:
   $p.key < q.key$

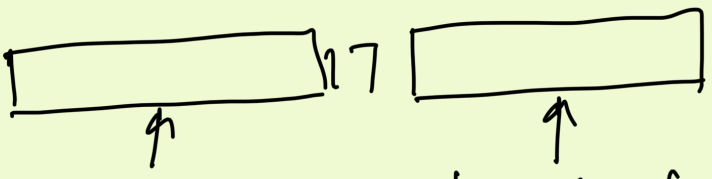


Examples

- The empty tree
- A tree with 1 node
- Every subtree of a binary search tree

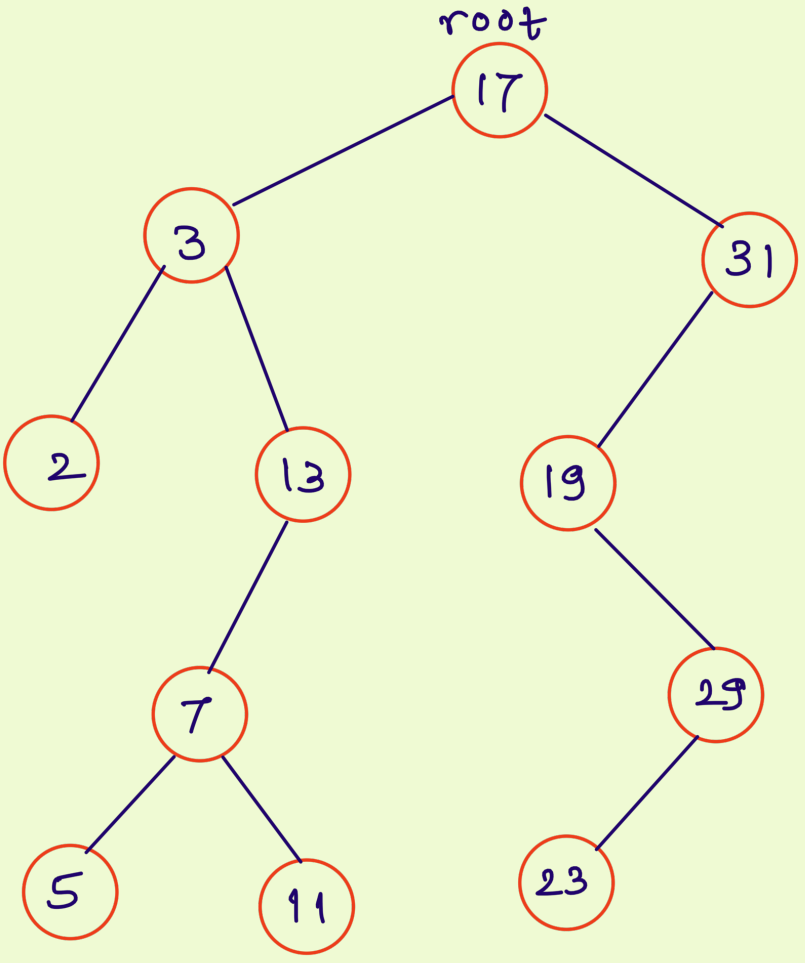


List Sorted



elements of left subtree of 17 in sorted order

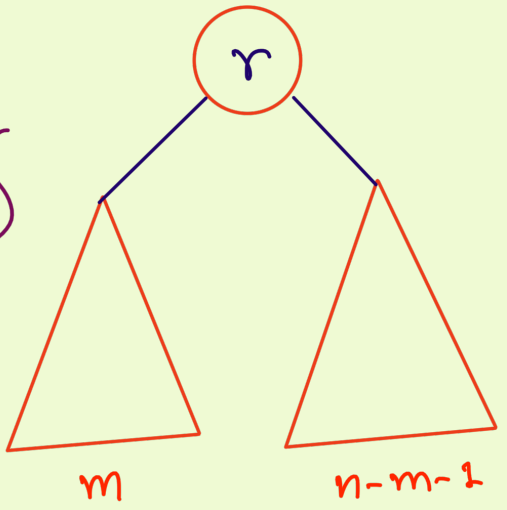
elements of right subtree of 17 in sorted order



List Sorted

```
List Sorted (r)
    If r is not None:
        List Sorted (r.left)
            Print (r.key)
        List Sorted (r.right)
```

(a.k.a. inorder traversal)



Running time: $T(0) = c_0$

$$T(n) = c_1 + \max_{0 \le m \le n-1} \left(T(m) + T(n-m-1) \right)$$

Prove that $\exists$ constants a, b such that $T(n) \le an + b$.

Corollary: $T(n) = O(n)$.

More Traversals

Pre-order

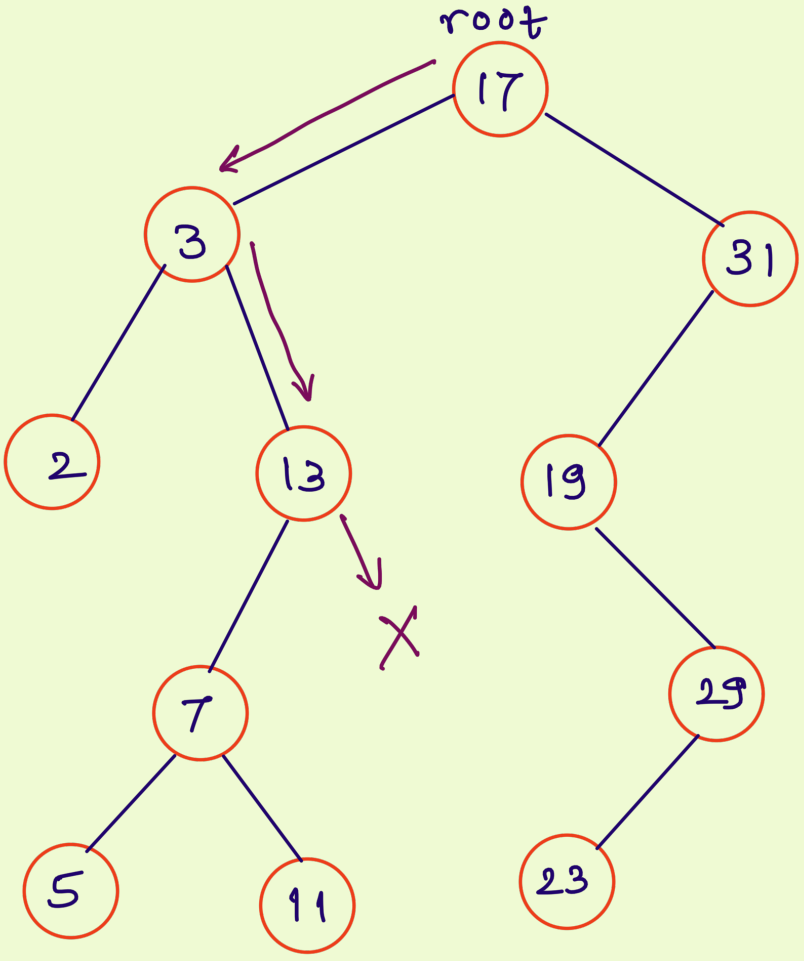
```
PreOrder (r):
    If r is not None:
        Print r.key.
        PreOrder (r.left)
        PreOrder (r.right)
```

Post-Order

```
PostOrder (r):
    If r is not None:
        PostOrder (r.left)
        PostOrder (r.right)
        Print r.key.
```

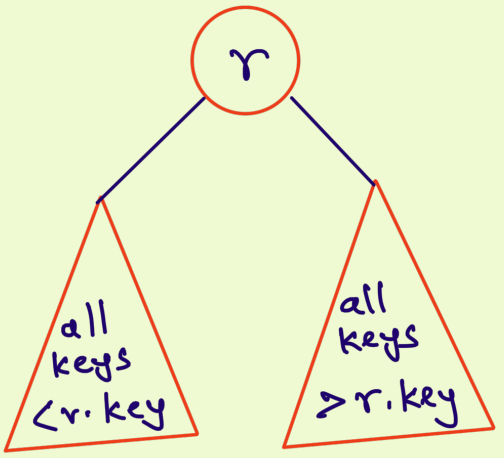
Searching

Search (15)



Search (x):

```
r ← root
while r is not None:
    if x = r.key:
        return TRUE
    if x < r.key:
        r ← r.left
    else
        r ← r.right
return FALSE.
```



$x \in$ tree iff $x \in$ left subtree of r

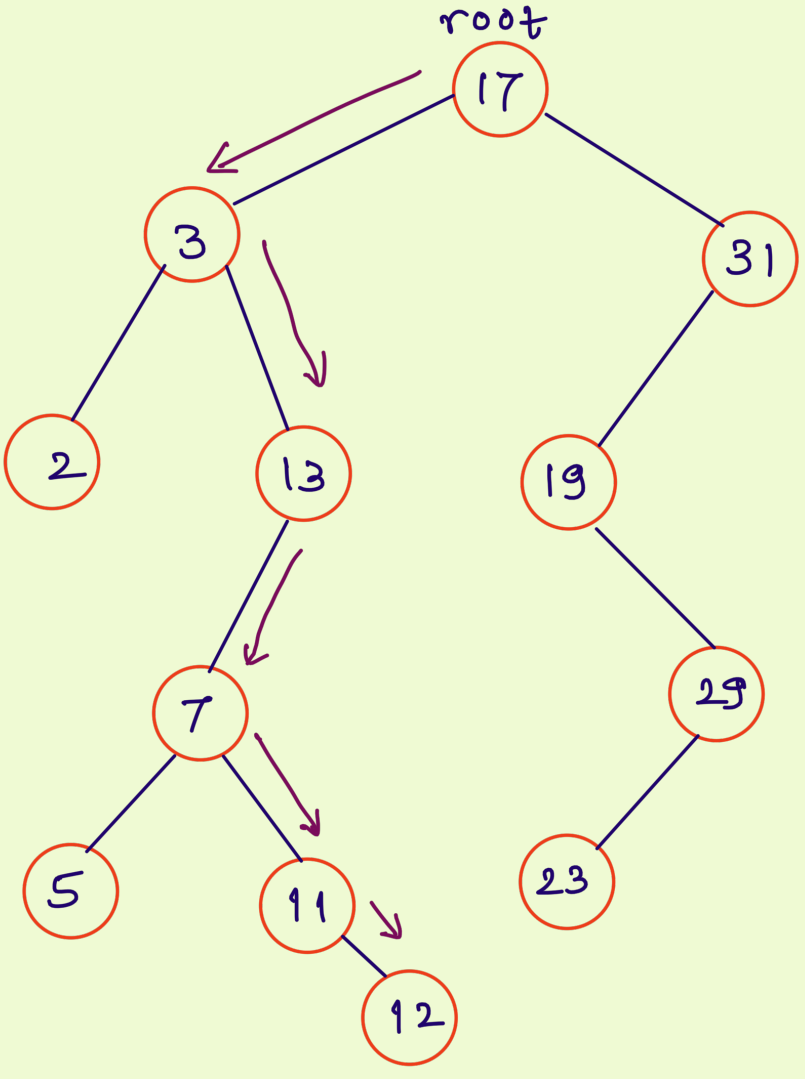
$x > r.key$

$x \in$ tree iff $x \in$ right subtree of r

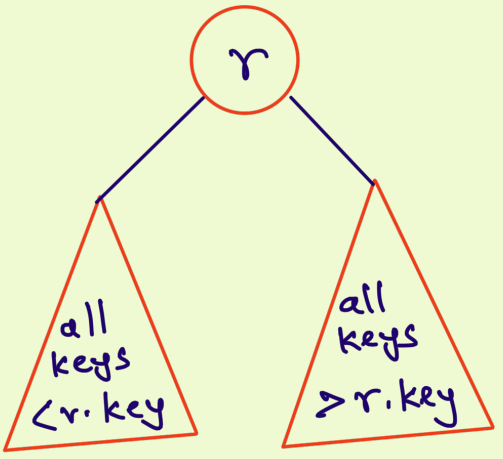
Running time: $O(h)$

Insertion

Insert (12).



Insert (x):



r ← root

Create a node q with
q.key ← x,
q.parent ← None,
q.left ← None,
q.right ← None.

If root is None:
root ← q and return

while TRUE:
If x = r.key
Throw Exception / Return

If x < r.key
If r.left is None
Make q the left child of r, and return
Else
r ← r.left.

Else (x > r.key)
If r.right is None
Make q the right child of r, and return
Else
r ← r.right

Exercise: Prove that the insertion operation maintains the binary search tree property.