Date: 22, 27 April 2021

Black Board

Data Structures

Lecture # 13 -- 14

Topics:

Trees & Search Trees

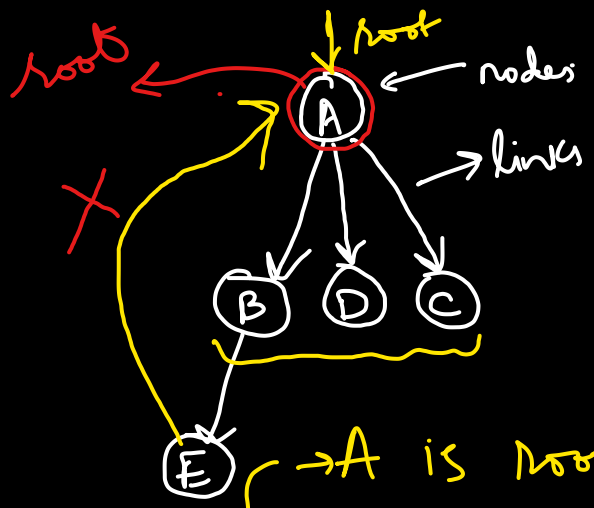
Binary Trees

Binary Search Trees

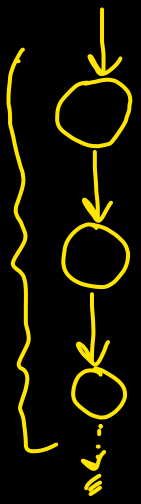
(1) Trees: Collection of nodes (vertices) and links (edges, hops, ptrs) (directed)

Properties

with certain properties. → tree has no cycle.



- Our trees will be rooted
- → There will be a unique root node.
- → Because of root we get a hierarchy.



→ A is root (In trees we use the language of family hierarchy)

→ A is the parent of B, D & C.

→ B, D & C are children of A.

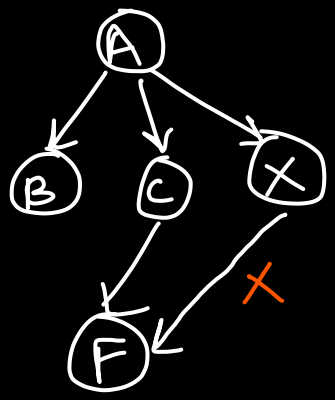
→ E is a grand child of A.

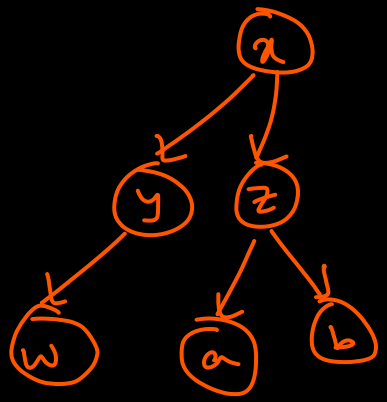
- B, D & C are siblings.

Another property of a rooted tree:

There is a unique path from the root to every node in the tree. (in other words each node has a unique parent).

Path: a sequence of hops & nodes between A & B → (A, B)





Nodes have 2 categories

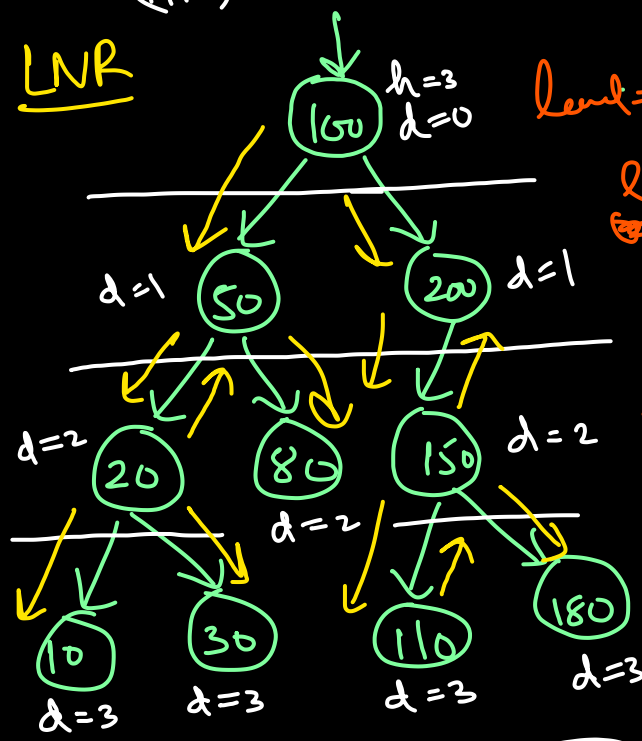
1. Internal Nodes: have one or more children
   - ↳ special internal node: root has no parent.
2. Leaf Nodes: have no children.

(2) Some special metrics for trees

(i) — height of a node
(ii) — depth of a node
(iii) — levels of a tree

height(h): is the length of the longest path from the node to a leaf node.

LNR



height(50) = 2
height(200) = 2
height(110) = 0 → leaves have height 0
height(100) = 3 → height of the root node is also the height of the tree.

Length of the path is the # of links on that path.

(ii) **Depth**: of a node is the length of the path from the root to that node. (distance from root)

10, 20, 30, 50, 80, 100, 110, 150, 180, 200 .....

(iii) Level: all node with the same depth are at (generation) the same level.

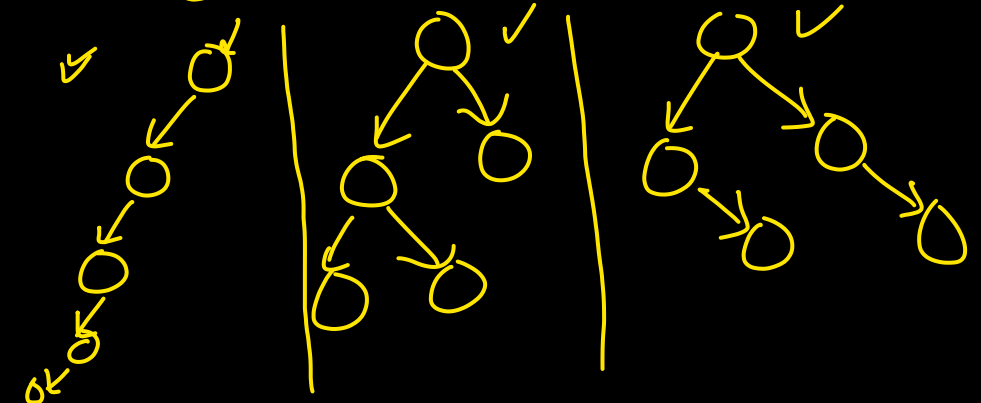
(3) Structural variations in a tree

→ we can bound the # of children.

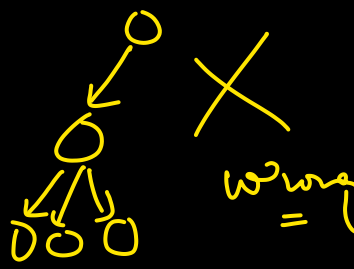
→ e.g. A tree where any node has 0, 1 or 2 children is called a binary tree.

→ we focus on the structure of a binary tree.

5 node binary tree



So on....



not in our discussion

forest

→ Height of a binary tree

$n \geq 0$

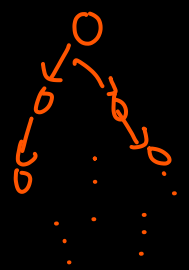
Question: Consider a binary tree over n nodes.

① → what could be its maximum height?

② → what could be its minimum height?

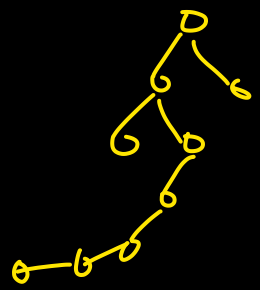
Ans: ① → n nodes can have max height $n-1$

i.e $O(n)$



⇒ So: a tree of a million nodes has height of million. ($10,00,000 \simeq 2^{20}$)

→ Completely Skewed tree



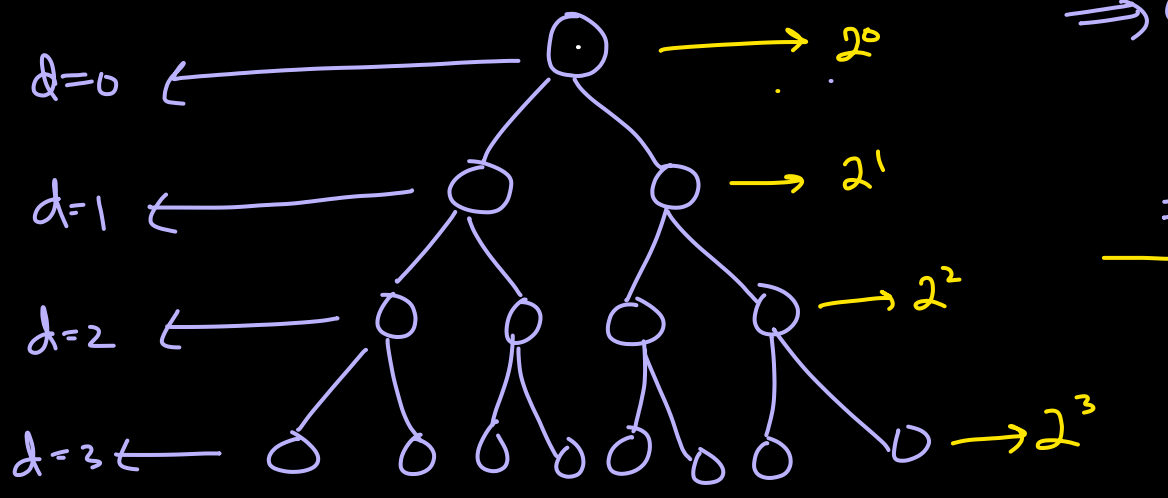
⇒ Skew increases height.

⇒ For a skewed tree $h = O(n)$

Ans 2 min height:

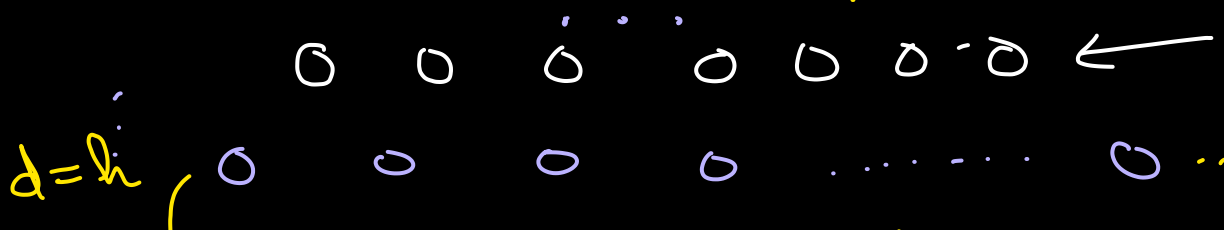
→ Every level can "eat up" a lot of nodes.

→ If every level is packed: → Perfect Binary Tree (all levels are full)

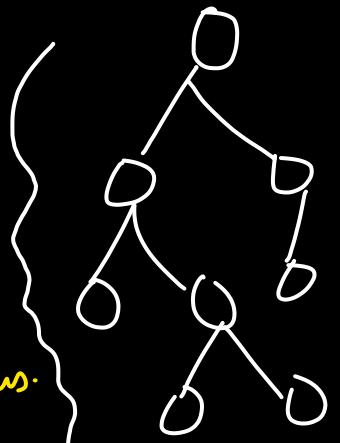


⇒ d=k level has at most how nodes

$= 2^k$ nodes.



↳ what is the value of d=?



⇒ we need to give balance precisely.

total # of nodes $= n =$ (# of nodes on level 0) + (# of nodes on level 1) + ⋮ + (# of nodes on level h)

$$\Rightarrow n = 2^0 + 2^1 + 2^2 + \cdots + 2^h$$

geometric sum: $S_k = \frac{a(r^k - 1)}{r - 1}$, $r = 2$, $a = 1$

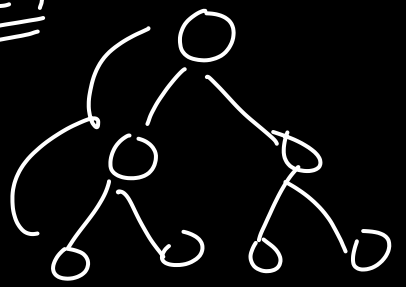
$$\Rightarrow n = 2^{h+1} - 1$$

$$2^{h+1} = n + 1$$

$$h + 1 = \lg(n+1)$$

$$\Rightarrow \boxed{h = \lg(n+1) - 1} = O(\lg n)$$

$n = 7$



1 million nodes $\approx 2^{20}$

$\log(7+1) - 1$
$= 3 - 1 = 2$

For a perfect tree of $2^{20} = n$
what is the height? Just 20!

---

lesson: $c \log n \leq h \leq cn$

Perfect Binary tree is the most balanced

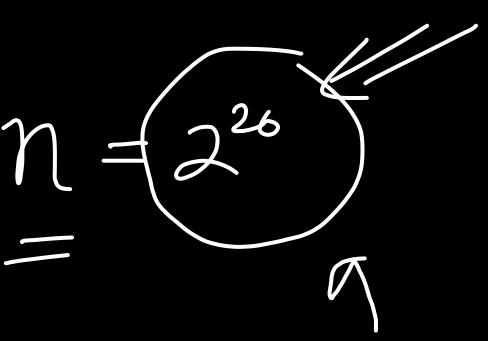
Balanced Binary tree

Skewed.

(C) A search tree will also use to search, insert & erase data based on a key.

↳ we will discuss a binary search tree:

Hint: Time for search will depend on height.



Lecture 14

Height Balanced Tree: is such a Tree whose height $h = O(\lg n)$ $\Leftarrow$ goal

---

Binary Search Tree

It has two properties:

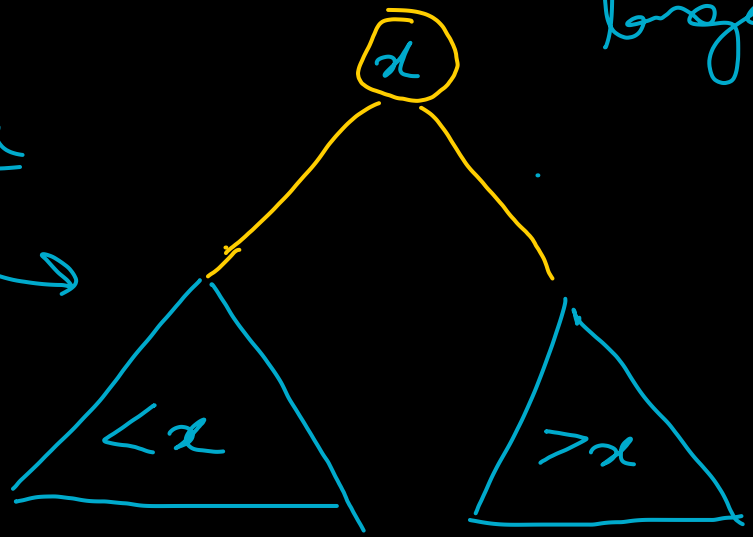
1. Structure property: it should be a binary tree.

2. Search property: (Defined below)

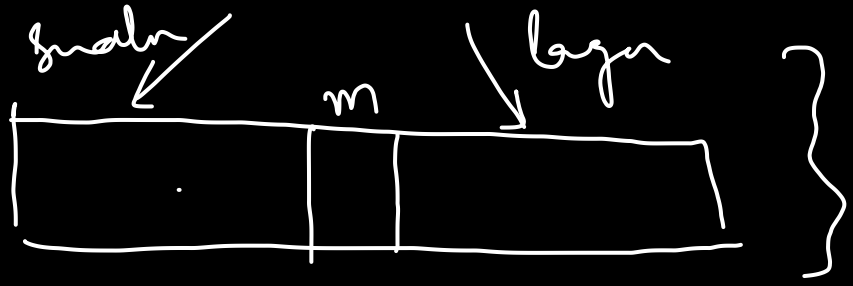
For any node x, the left subtree of x contains smaller keys than and the right subtree contains larger keys than x.

(unique keys)

left sub-tree



- Designed to mimic the behavior of binary search.



Binary Search

$O(\log n)$

Operations on a BST (as a Dynamic Set)

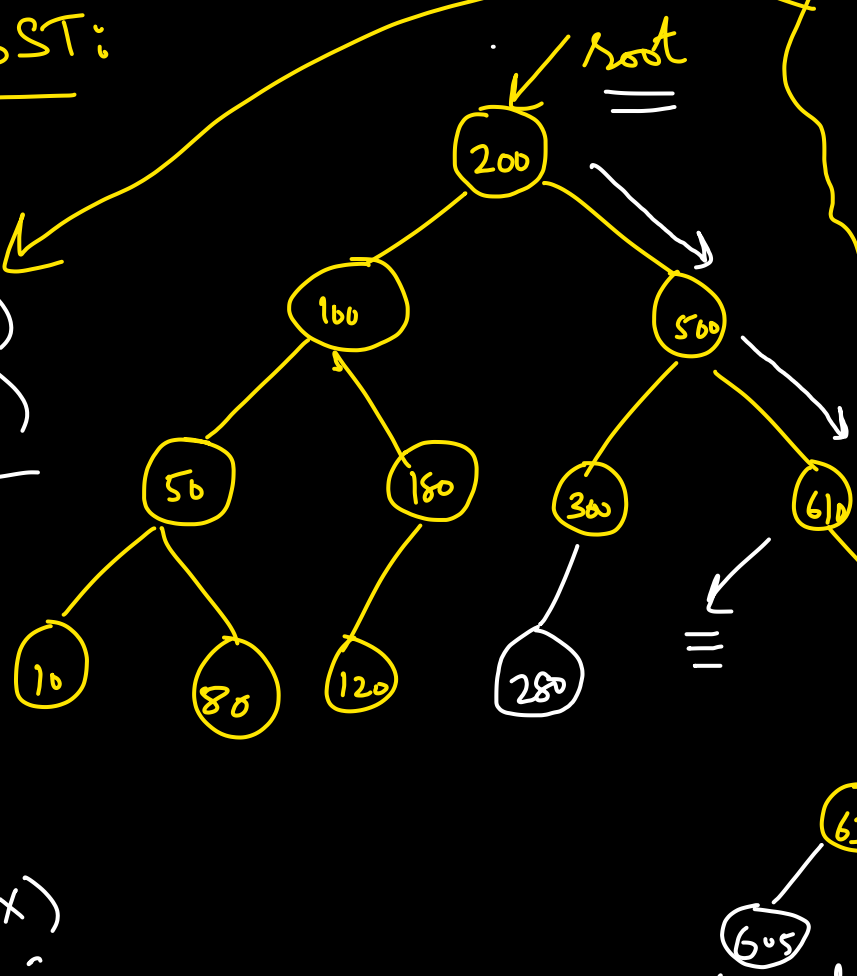
↳ size can change at run time

- ✓ Insert
- — Erase
- ✓ Search
- — Housekeeping: copy const, Dest, ...

We must maintain both the structure and search properties while executing these operations.

— print ⟹ prints the data in ascending order of the keys.

Example BST:



Successor & predecessor

x y

ordered keys.

x is the predecessor of y

y is the successor of x

Successor(x)
Predecessor(x)

we must write these.

(i) Search(x):

- Start from root as current node
- If the current node is x, return success
- else if curr node is null return failure.

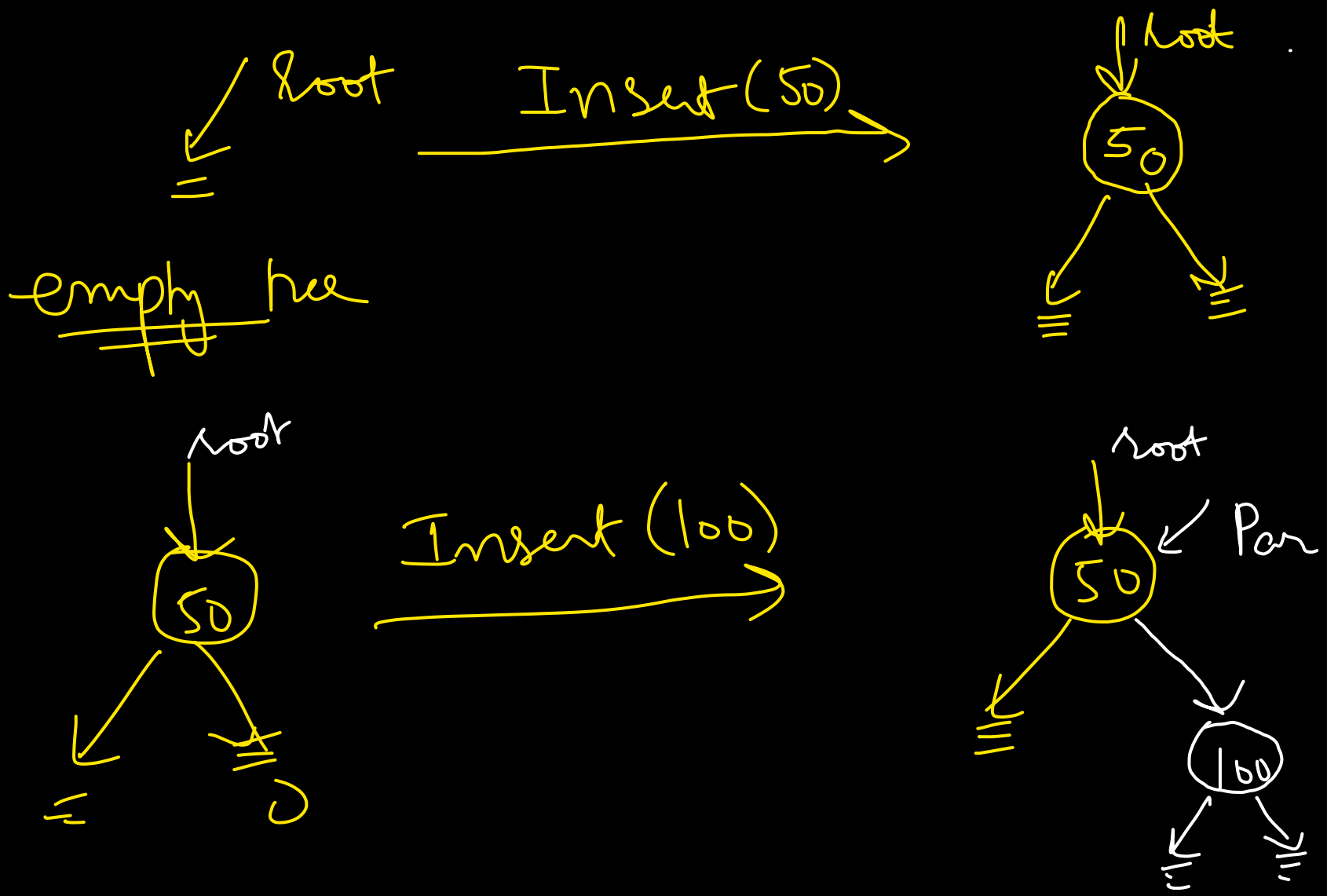
- else if x is smaller than the curr node data, move to the left subtree

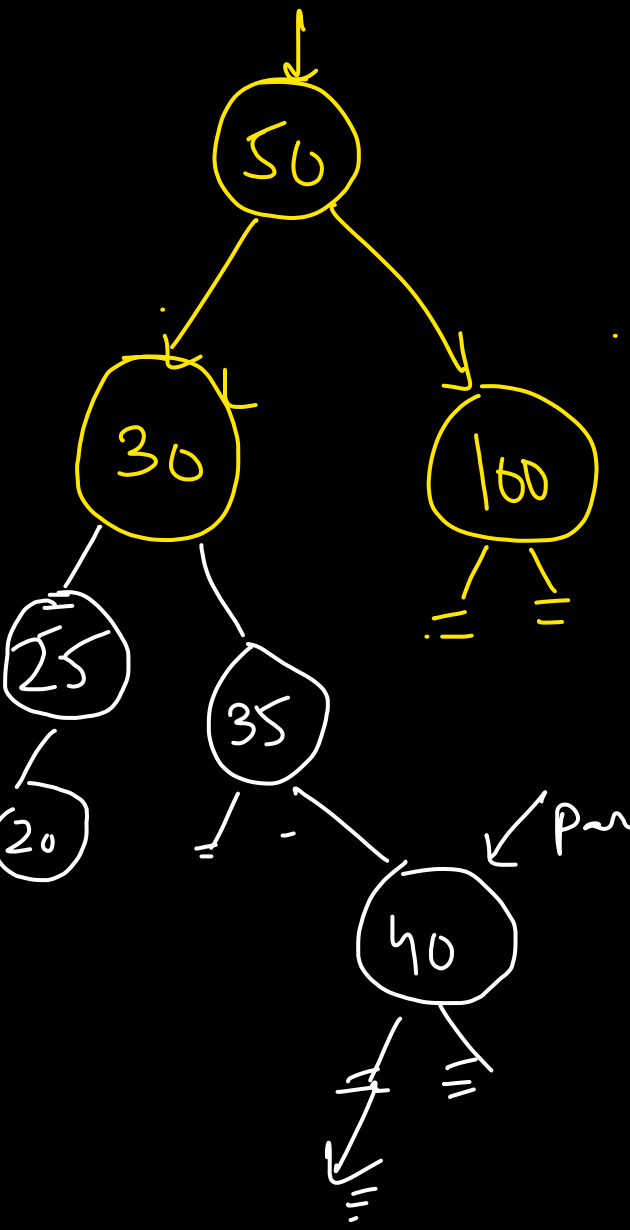
else if x is bigger than the curr node data, move to the right subtree.

$T(n)$ = length of the longest path from root to a leaf

$T(n) = O(h)$ { remember $c \log n \leq h \leq dn$

(ii) Insert(x):


root
Insert(50)
root
50
empty tree
root
50
Insert(100)
root
Par
50
100



Insert(30)

Insert(35)

Insert(40)

Insert(25)

Insert(20)

$T(n) = O(h)$

② $c \log n \leq h \leq d n$

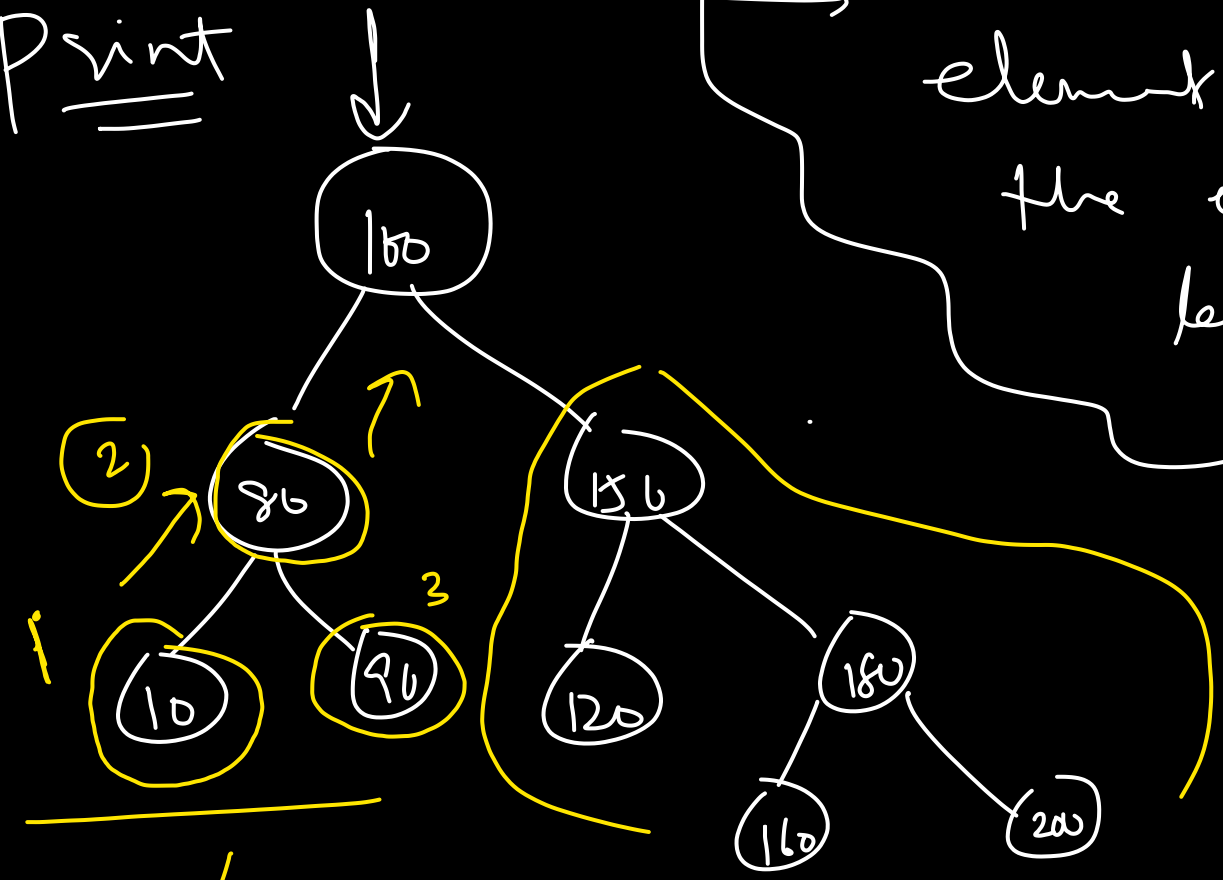
private:

search

```
bool getAccess(const T& Key,
               treeNode* & par);
```

Note: Every incoming node in a bst becomes a leaf node.

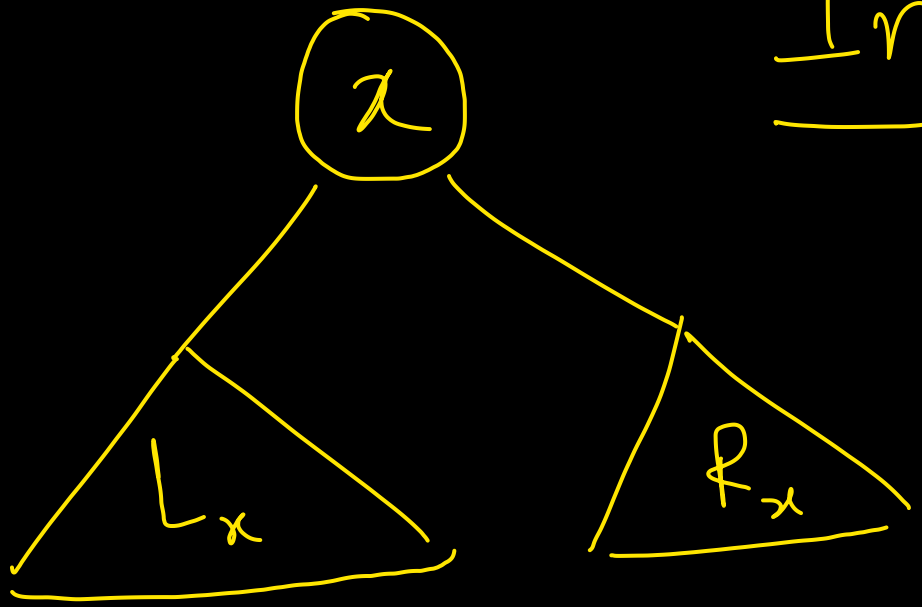
(iii) Print → the smallest element is on the extreme left.



→ left child, parent, right child.

10, 80, 90 → low → Recursion

In order Traversal



LNR

left, right node