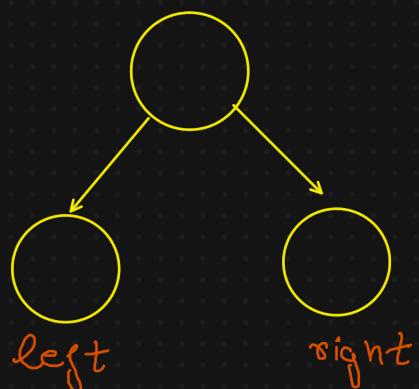


Binary Tree

a binary tree is a specific / specialized kind of generic tree which can have atmost 2 children.

0, 1 or 2 children allowed



→ makes the structure a lot simpler than generic trees

→ extremely useful for various computational purposes.

Real-life examples :-

- Yes / No Decision making
- Tournament Structure
- family tree with 2 parent

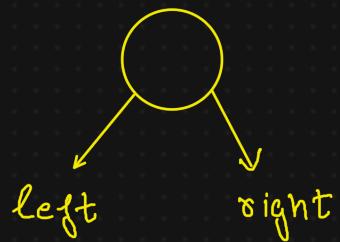


Industry examples :-

- Binary DT in AI / ML
- Database Searching
- Binary trees in compiler

Binary Tree Node

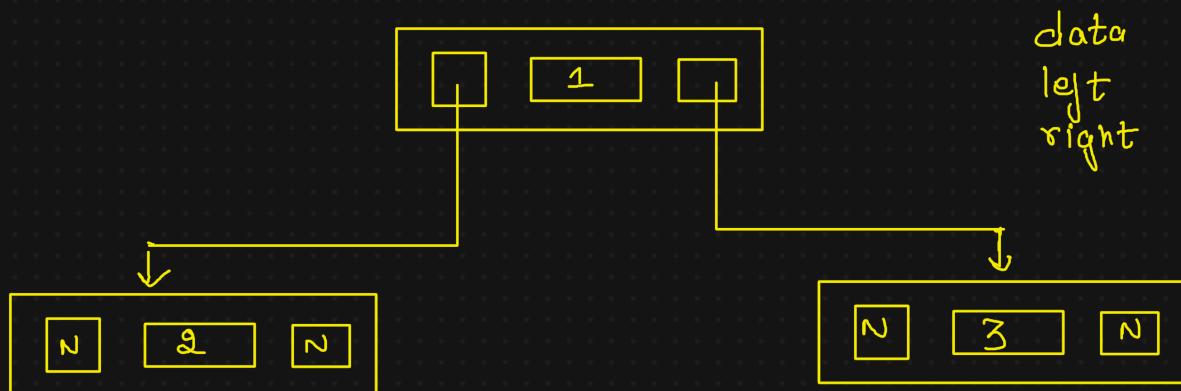
data
children = []



→ Each of left and right node is going to be a Binary Tree



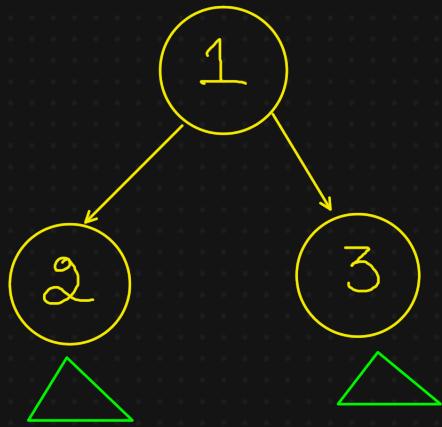
\neq
not equal



Print Binary Tree

→ recursion can be used as each node is a sub-binary tree

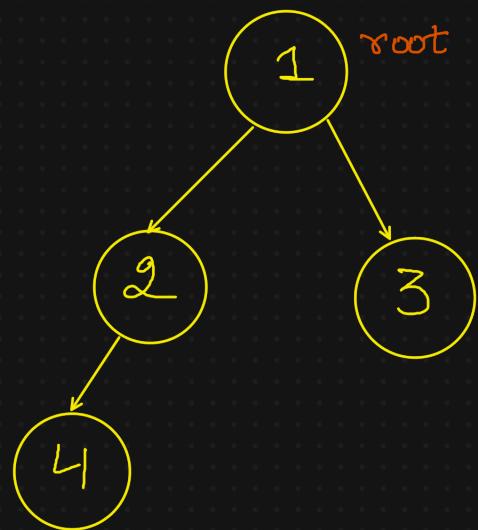
1: L → 2 R → 3
2: L → None R → None
3: L → None R → None



Take Input of a Binary Tree

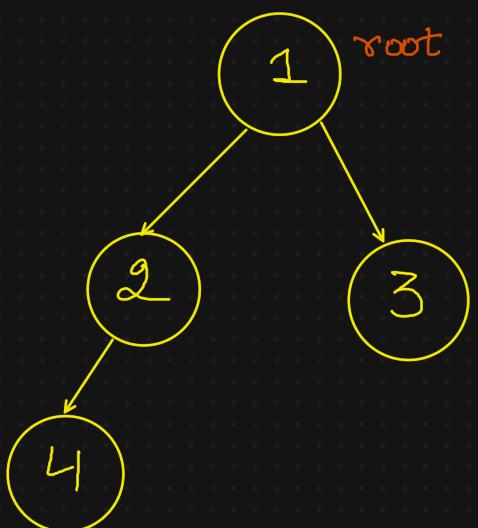
Level wise

3 2 1



Point Tree level wise

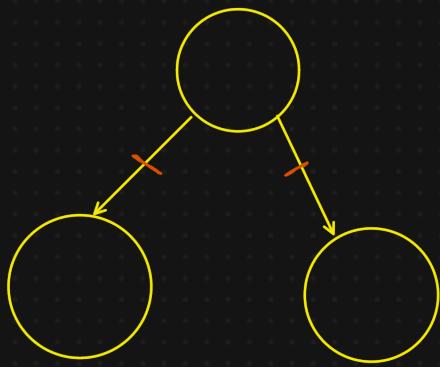
1 : L:2 R:3
2 : L:4 R:N
3 : L:N R:N
4 : L:N R:N



Diameter of a tree

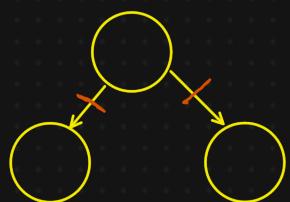


maximum distance between
two points



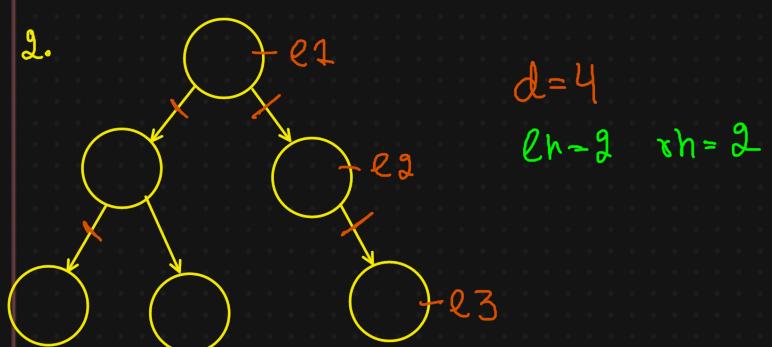
maximum distance between
2 nodes

1.



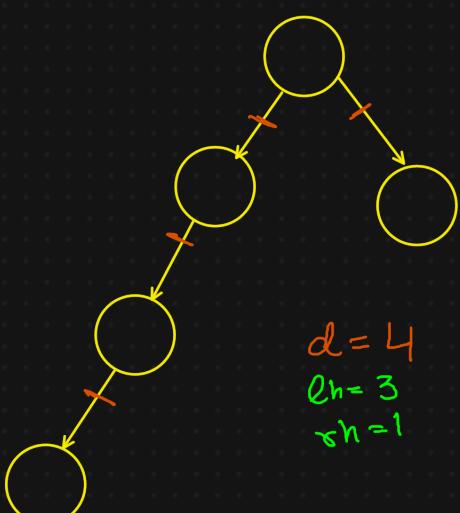
$$d = 2 \\ \ell h = 1 \times h = 1$$

2.



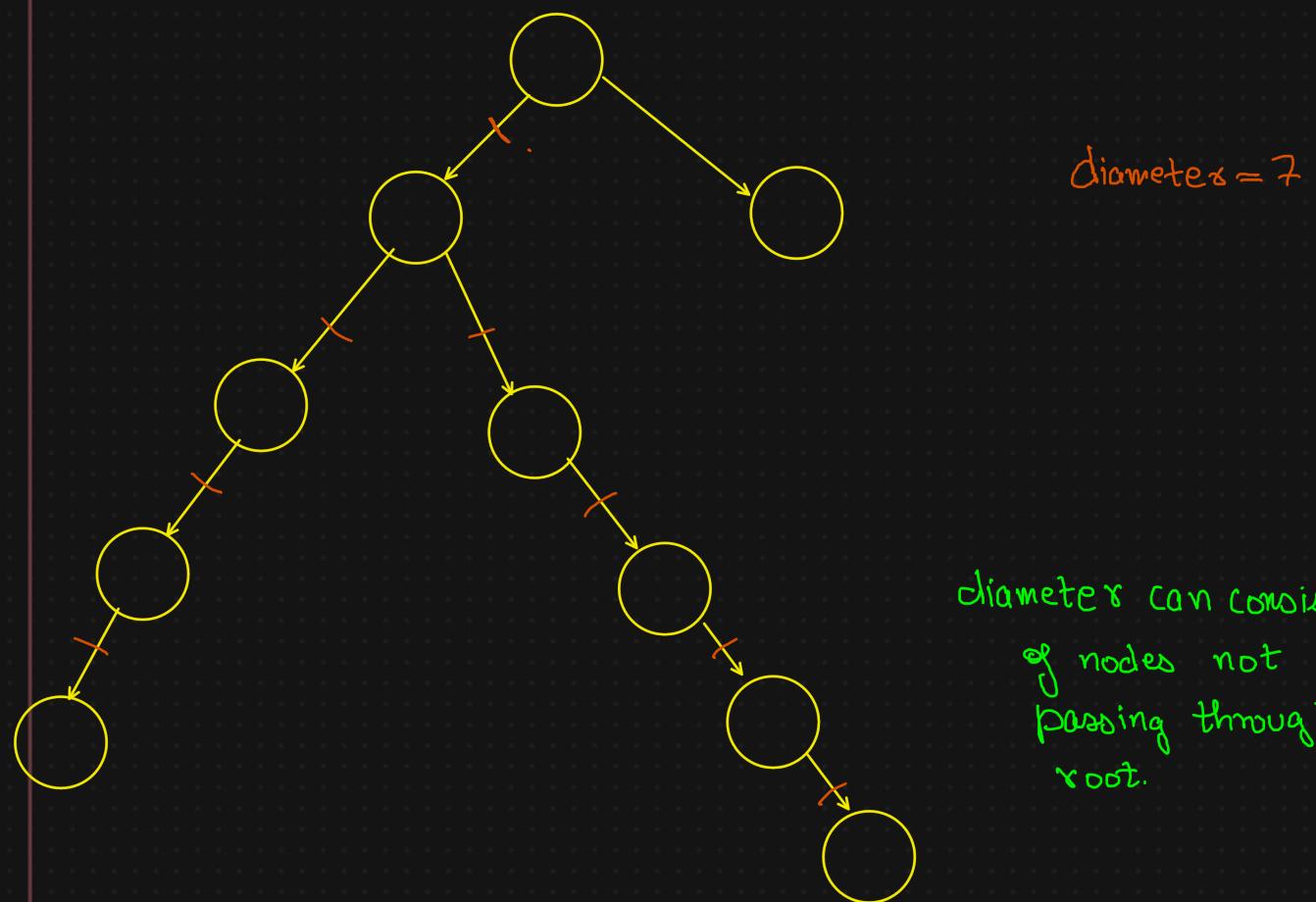
$$d = 4 \\ \ell h = 2 \times h = 2$$

3.



$$d = 4 \\ \ell h = 3 \\ \tau h = 1$$

\times diameter of a tree = left height + right height
wrong assumption



diameter can consist
of nodes not
passing through
root.

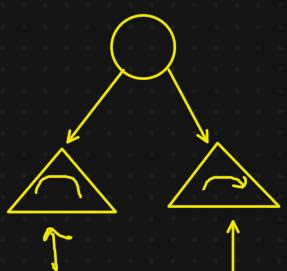
1. diameter through root

$$\ell_h + r_h = \text{diameter}$$

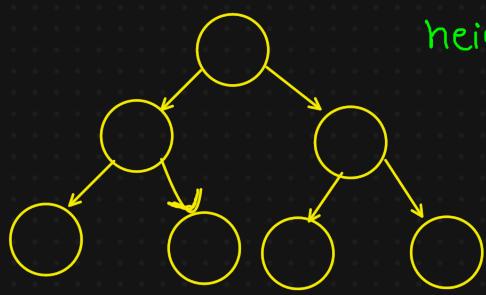
2. left diameter

3. right diameter

maximum of all above is my diameter



Complexity of height function is $\mathcal{O}(n)$



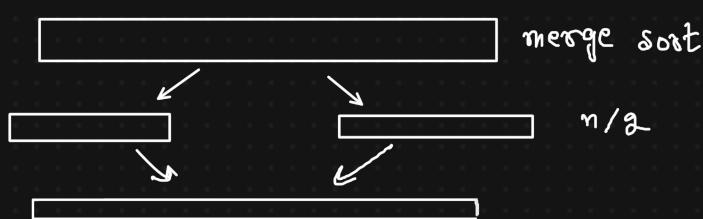
height = $\log n$

$$T(n) = \underbrace{Rn}_{\text{height}} + \overbrace{R}^{\mathcal{O}} + \underbrace{2T(n/2)}_{\text{diameter}}$$

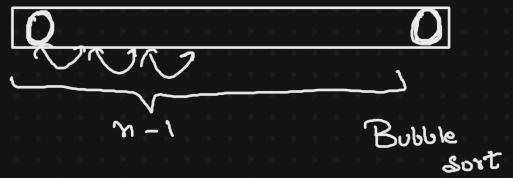


height = n

$$T(n) = \underbrace{Rn}_{\mathcal{O}} + \overbrace{R}^{\mathcal{O}} + T(n-1)$$



$= \mathcal{O}(n \log n)$

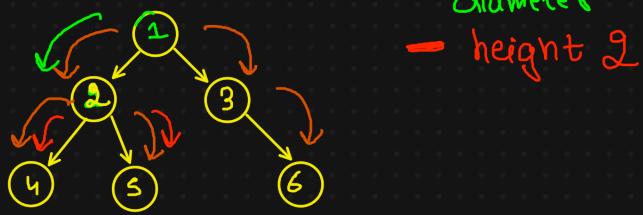


$\mathcal{O}(n^2)$

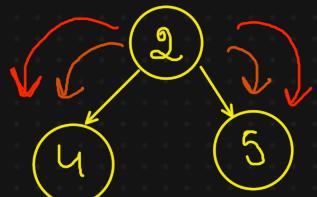
$\mathcal{O}(n \cdot h)$ height
No. of nodes height

Diameter of tree - Optimized

- height
- diameter
- height 2

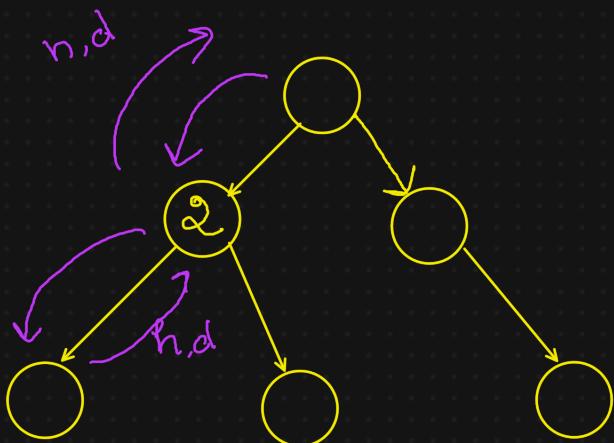


for 2, to get the diameter



2 calls for some information

1. height
 2. diameter
- } $\Rightarrow h, d$



instead of just the height
we will ask children
to get both height
and diameter.

$O(n)$

Final complexity

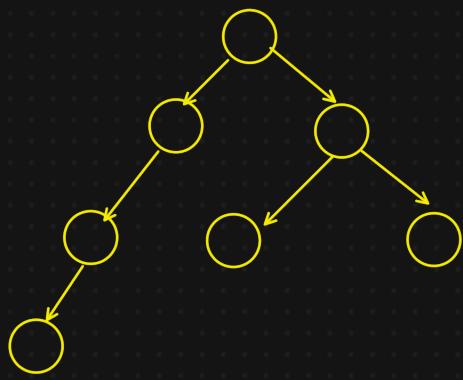
Balance Binary Tree

if the given binary tree is balanced or not?

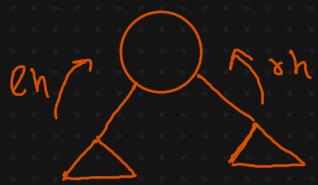
If at each node,

$$|lh - rh| \leq 1$$

that our tree is height balanced.



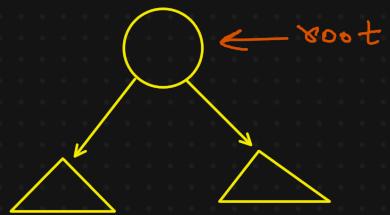
1st)



$$lh - rh \leq 1$$

: s-balanced (left-node)

: g-balanced (right node)

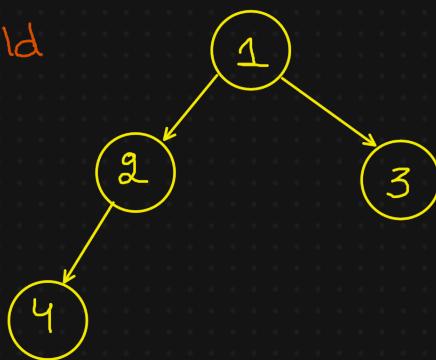


Traversal in Binary Trees

1. Preorder: first parent and then child

1 2 4 3

Always we call left first
and then right



2. Postorder: first my children and then me
child → parent

4 2 3 1

3. Inorder traversal: first the left , then me , then right
left child → parent → right child
4 2 1 3

1. Preorder : root left child, right child,

2. Postorder : left child right child root

3. Inorder : left child root right child,

Construct a tree from inorder and preorder

inorder : left subtree root right subtree

4 2 5

1

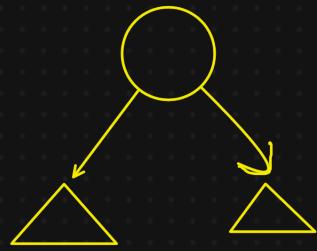
3 6

preorder: root left subtree right subtree

1 2 4 5

8I 8I+1

3 6



Preorder = 1 2 4 5 3 6

Post Order = 4 5 2 6 3 1

Inorder Traversal = 4 2 5 1 3 6

inorder : 4 2 5

preorder: 2 4 5



3 6

3 6

inorder : 4

preorder: 4



5

5



6

6



inS inE
inorder : left subtree root right subtree

4 2 5 1

inS

inS inE
right subtree

3 6

index

preorder: root left subtree right subtree

1 2 4 5

preS

3 6

preE

inS \rightarrow inorder

root

index

Construct tree [inorder, preorder, inS, inE, preS, preE]

left l inS = inS l inE = inS - 1 l prS = preS + 1 l prE

right r inS = inS + 1 r inE = inE r prS = l prE + 1 r prE = preE

$l prE - l prS = l inE - l inS$ $r prE - r prS = r inE - r inS$

Construct tree from preorder and Inorder - Solution

Construct a Binary Tree given inorder and postorder

inorder : 4 2 5 $\underset{\text{in} \times I}{1}$ 3 6
left subtree root right subtree

postorder : 4 5 2 6 3 1
left subtree right subtree root

root: postorder [n-1]

$$\text{left: } l_{\text{inS}} = \text{in} \times \emptyset$$

$$l_{\text{inE}} = \text{in} \times I - 1$$

$$l_{\text{poS}} = \text{po} \emptyset$$

$$l_{\text{poE}} = l_{\text{poS}} + (l_{\text{inE}} - l_{\text{inS}})$$

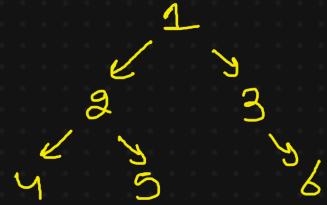
$$\text{right: } r_{\text{inS}} = \text{in} \times I + 1$$

$$r_{\text{inE}} = \text{in} E$$

$$r_{\text{poS}} = l_{\text{poE}} + 1$$

$$r_{\text{poE}} = \text{poE} - 1$$

$$l_{\text{inE}} - l_{\text{inS}} = \\ l_{\text{poE}} - l_{\text{poS}}$$



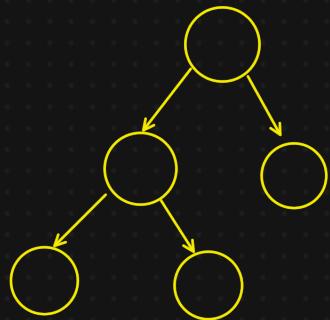
Post Order = 4 5 2 6 3 1
Inorder Traversal = 4 2 5 1 3 6

Types of Binary Tree

1] On basis of number of children

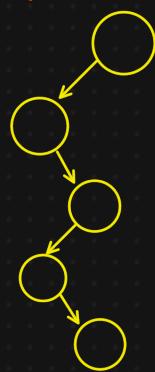
(a) Full Binary Tree

a type of binary tree in which every parent/internal node has either two children or no children.
1 child not allowed.



(b) Degenerate Binary tree

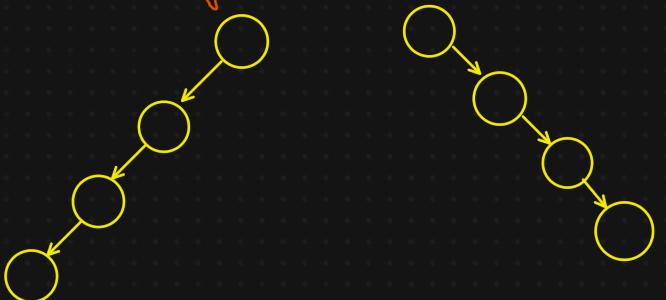
a tree in which every internal node has just one child either left or right. \Rightarrow



analogous to linked list in terms of operation.

c) Skewed Binary tree

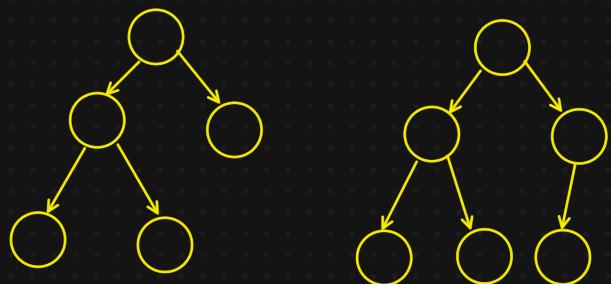
a degenerate binary tree in which the tree is dominated by either the left node or the right node.



2) On basis of completion of level.

a. Complete Binary tree [CBT]

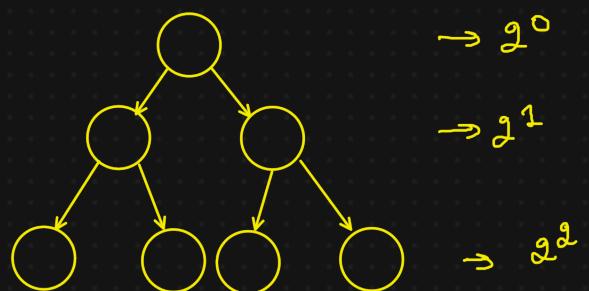
A binary tree in which all the levels are completely filled except possibly the last level and the last level has the nodes as to the left as possible.



b. Perfect binary trees

A binary tree in which all the internal nodes have 2 children and all leaf nodes are at the same level.

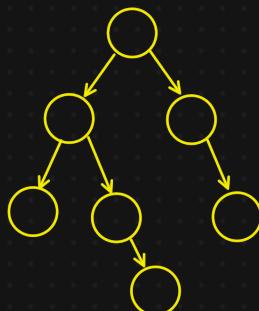
$$\text{Leaf nodes count} = \frac{1}{2} + \text{internal nodes}$$



$$\begin{aligned} \text{Leaf nodes count} &= 2^{h+1} - 1 \\ 2^{h+1} - 1 &= \frac{8-1}{7} \end{aligned}$$

c. Balanced Binary tree

A binary tree in which at every node height of left and the right subtree for each node is either 0 or 1.



Binary Search tree : Special Kind of Binary Trees

A binary tree in which each node of tree follows the below property

→ left subtree contains node whose value is less than root's value

→ right subtree contains node whose value is greater than root's value

→ left and right subtree must also be Binary search tree

