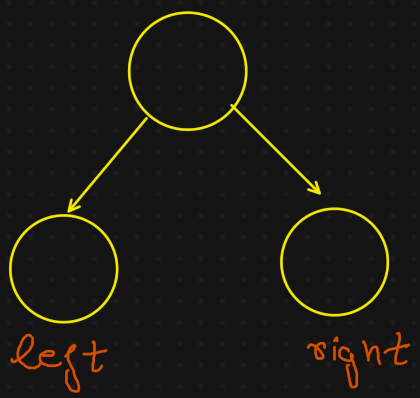
Binary Tree

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

0, 1 or 2 children allowed

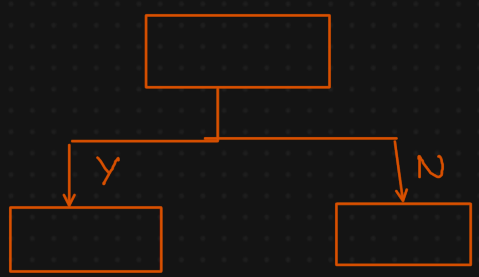


→ makes the structure a lot simpler than generic trees

→ extremely useful for various computational purposes.

Real-life examples :-

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



Industry examples :-

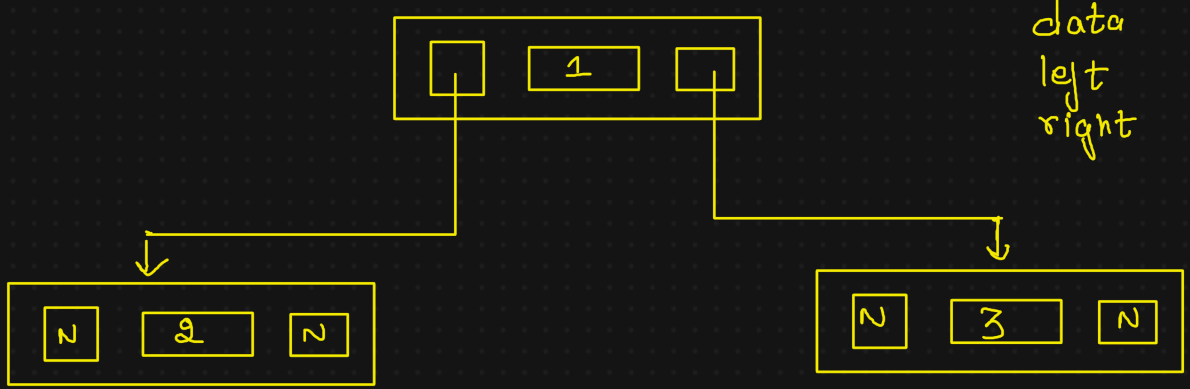
→ Binary DT in AI / ML
→ Database Searching
→ Binary trees in compiler

Binary Tree Node

data
children = [ ]

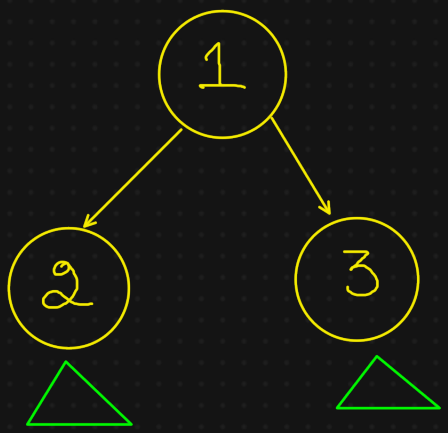
→ each of left and right node is going to be a Binary Tree

→



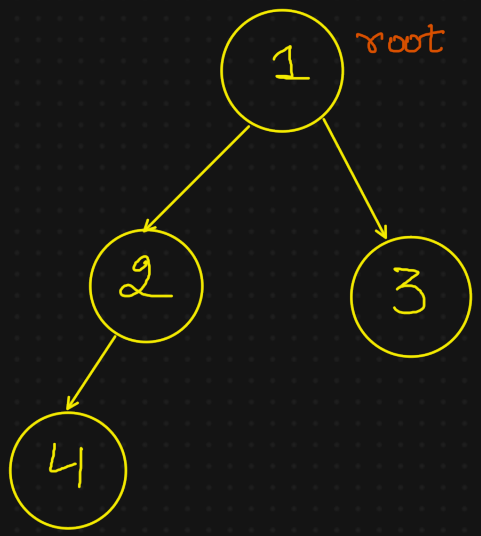
Print Binary Tree

→ 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

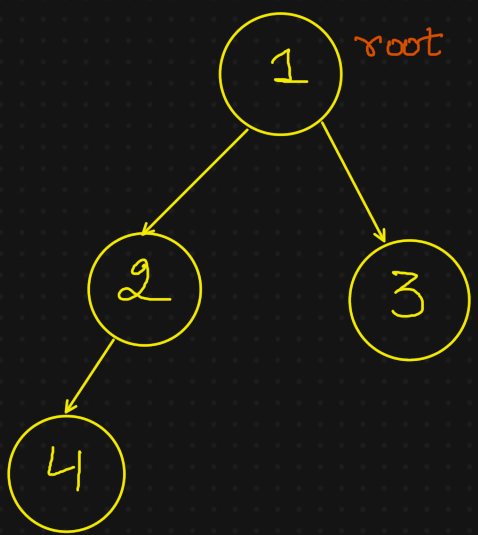
Take Input of a Binary Tree



Level wise

3 2 1

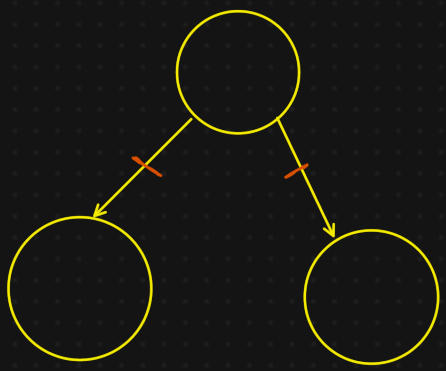
Print Tree level wise



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

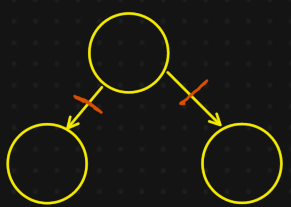
Diameter of a tree

maximium distance between two points



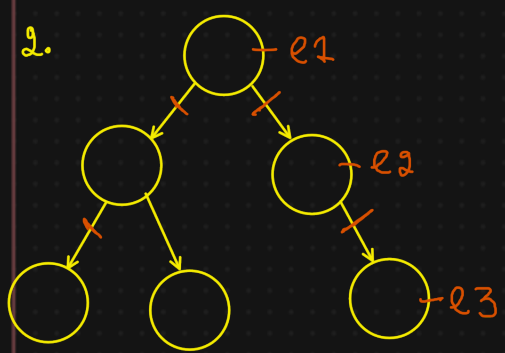
maximum distance between 2 nodes

1.



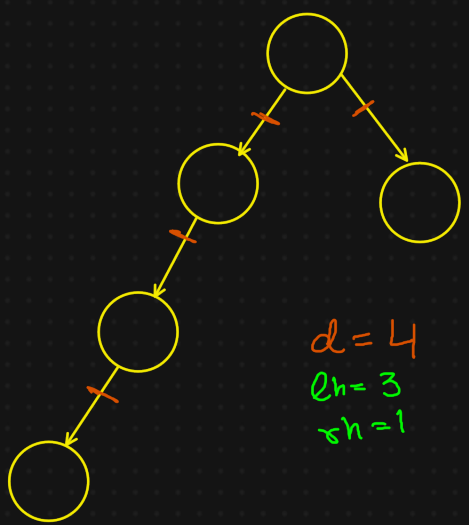
d = 2
lh = 1 rh = 1

2.



d = 4
lh = 2 rh = 2

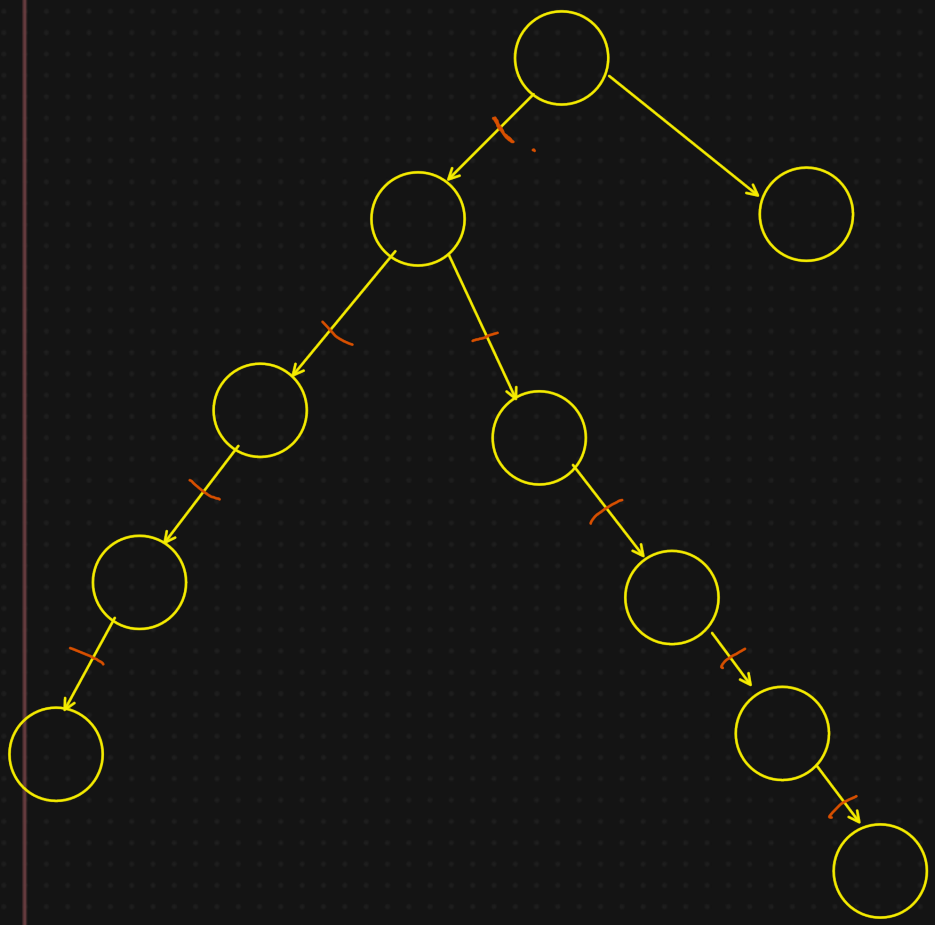
3.



d = 4
lh = 3
rh = 1

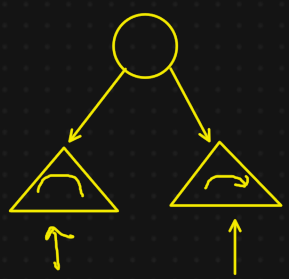
✗ diameter of a tree = left height + right height

wrong assumption



diameter = 7

diameter can consist of nodes not passing through root.



1. diameter through root

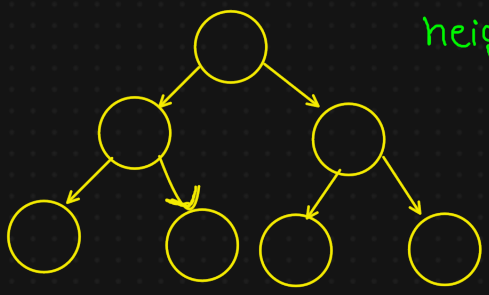
$lh_1 + rh_1$ = diameter

2. left diameter$_3$

3. right diameter$_4$

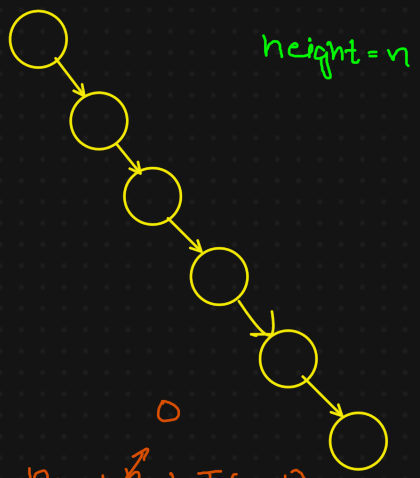
maximum of all above is my diameter

Complexity of height function is $O(n)$



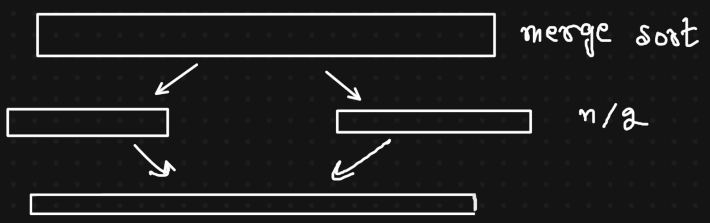
height = log n

$T(n) = \underbrace{Rn}_{\text{height}} + \cancel{R}^{0} + \underbrace{2T(n/2)}_{\text{diameter}}$

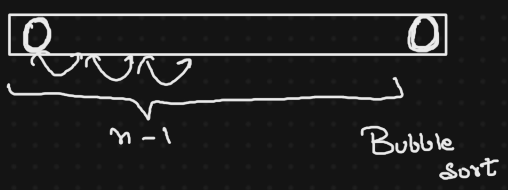


height = n

$T(n) = Rn + \cancel{R}^{0} + T(n-1)$



$= O(n \log n)$



$O(n^2)$

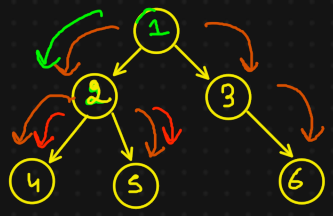
$O(n * \underbrace{h}_{\text{height}})$

↙ no. of nodes

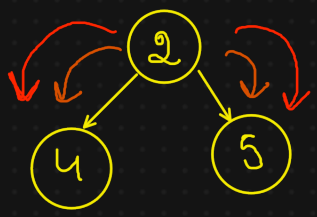
↘ height

Diameter of tree - Optimized

- height
- diameter
- height 2



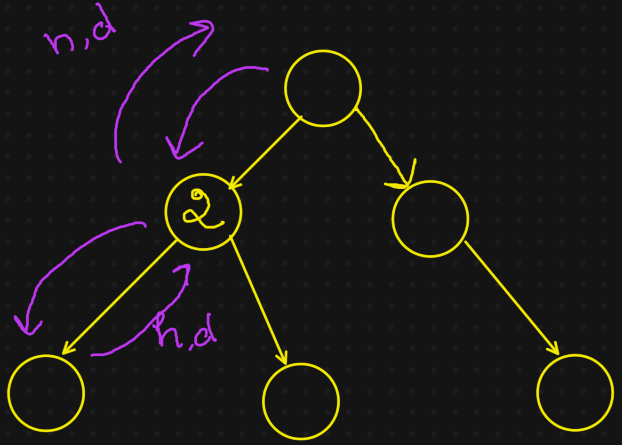
for 2, to get the diameter



2 calls for some information

1. height
2. diameter

} ⇒ h,d



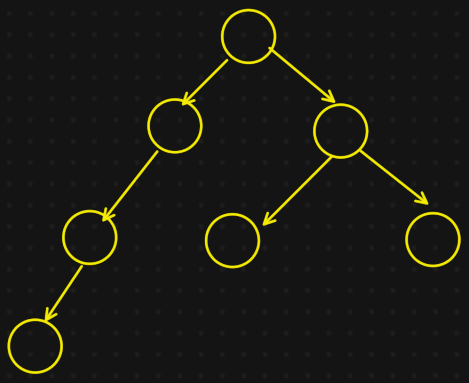
instead of just the height we will ask children to get both height and diameter.

O(n)

Final complexity

Balance Binary Tree

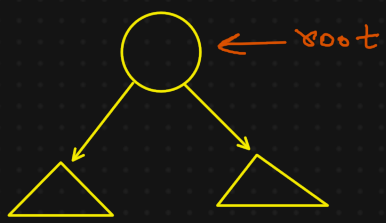
if the given binary tree is balanced or not ?



If at each node,

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

that our tree is height balanced.



1st)

$lh - rh \leq 1$

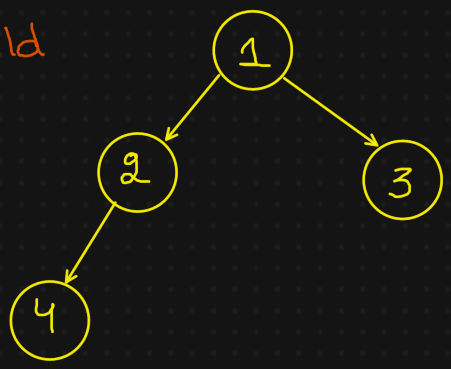
is_balanced (left_node)
is_balanced (right node)

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

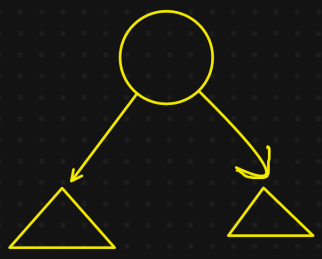
   4 2 1 3

1. Pre order: root left child right child

2. Post order: 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

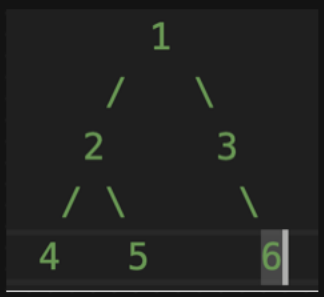
len

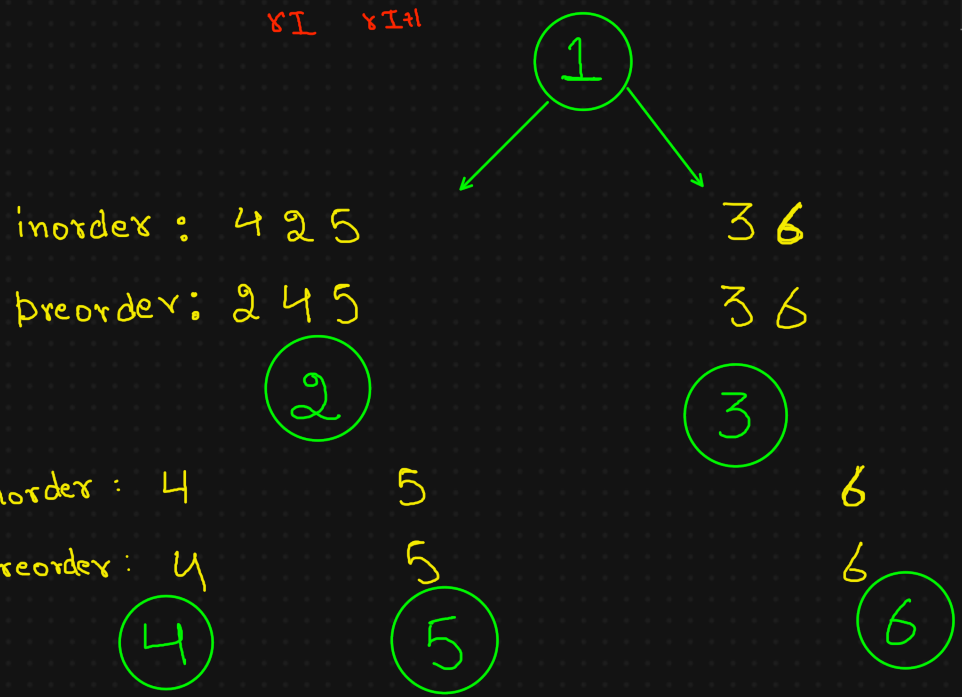
preorder: root | left subtree | right subtree

1 | 2 4 5 | 3 6

rI | rI+1

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





inS inE index

inorder : left subtree | root | right subtree

4 2 5 | 1 | 3 6

inrI

preorder: root | left subtree | right subtree

1 | 2 4 5 | 3 6

rI preS preE

inrI → inorder root index

Construct tree (inorder , preorder , inS , inE , prS , prE)

left: linS = inS | linE = inrI - 1 | lprS = preS + 1 | lprE

right: rInS = inrI + 1 | rinE = inE | rprS = lprE + 1 | rprE = prE

lprE - lprS = linE - linS | rprE - rprS = rInE - rinS

Construct tree from preorder and Inorder - Solution

Construct a Binary Tree given inorder and postorder

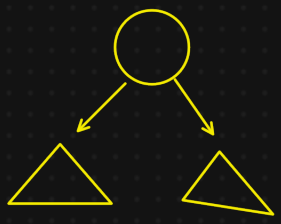
inorder : 4 2 5 1 3 6

inI

left subtree | root | right subtree

postorder : 4 5 2 6 3 1

left subtree | right subtree | root



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

root : postorder [n - 1]

left :

linS = inS

linE = inI - 1

lpoS = boS

lpoE = lpoS + (linE - linS)

linE - linS = lpoE - lpoS

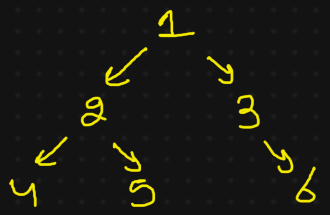
right :

rinS = inI + 1

rinE = inE

rpoS = lpoE + 1

rpoE = poE - 1



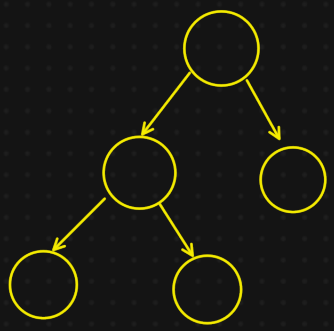
Types of Binary Tree

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.

analogous to linked list in terms of operation.

⇒

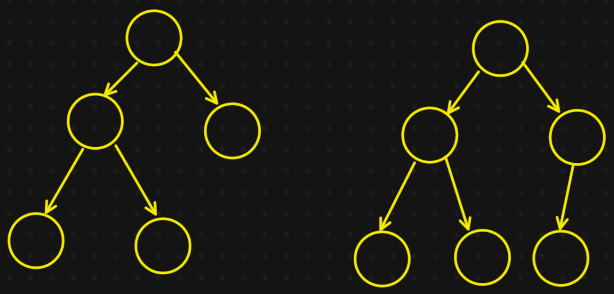
c) Skewed Binary tree

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]

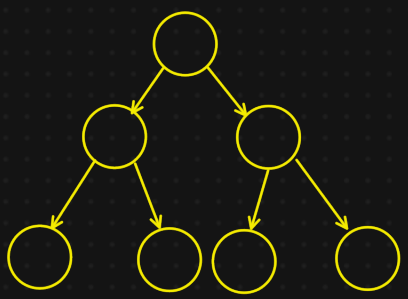
a binary tree in which all the levels are completely filled except possibly the last level and the last level has the nodes as to the left as possible.



b. Perfect binary trees

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

Leaf nodes count = 1 + internal nodes



$\rightarrow 2^0$

$\rightarrow 2^1$

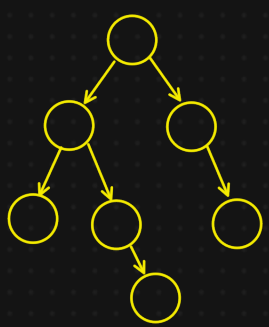
$\rightarrow 2^2$

$2^{h+1} - 1$

$2^{2+1} - 1 \Rightarrow 8 - 1 = 7$

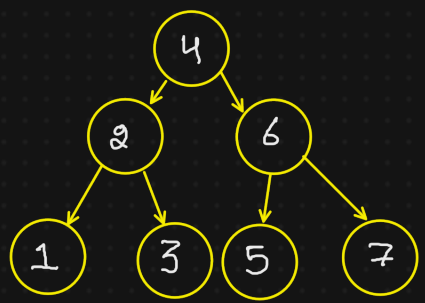
c. Balanced Binary tree

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