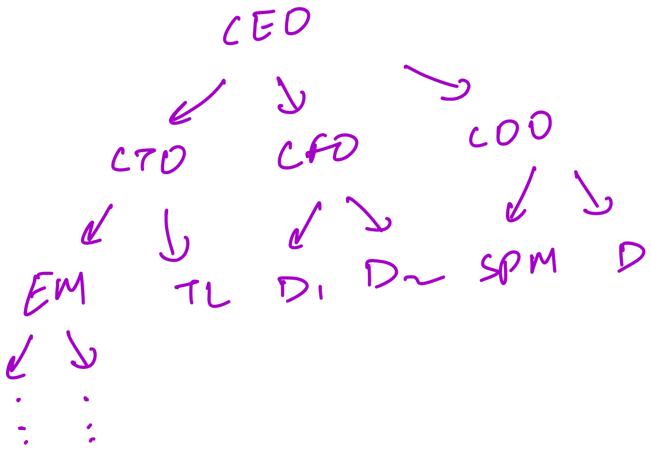
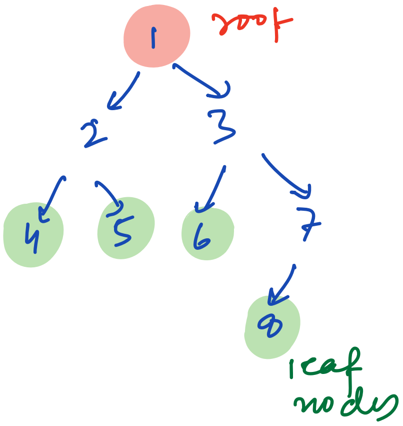
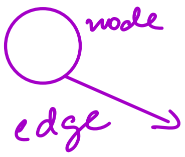
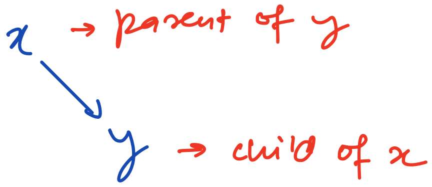
Trees 1 : Structure & Traversal

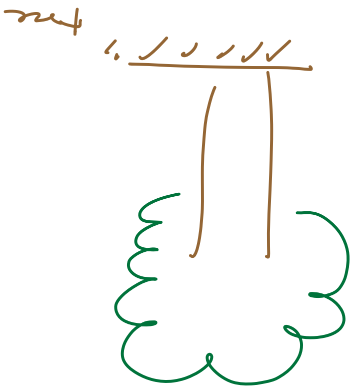
Hierarchial DS









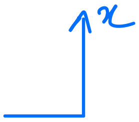


Root → topmost node of a tree, it is the tree representative.

↳ only node without parent

Leaf → node without children

Height → # edges to travel from node x to farthest leaf.

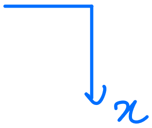


height(3) = 2 height(leaf) = 0

height (1) = 3

Height of tree = Height(root)

Depth/Level → # edges to travel from root to current node x.



depth(3) = 1 depth(8) = 3

depth(root) = 0

Siblings → Nodes that have same parent.

4 & 5 are siblings

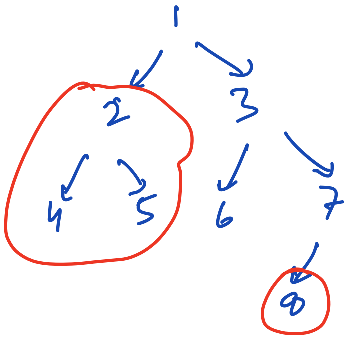
Ancestors → All nodes from parent to the root are ancestors.

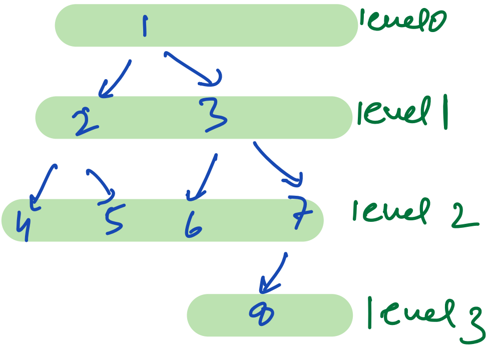
Nodes 7,3,1 are ancestors of 8.

Descendants → All nodes from children to leaf.

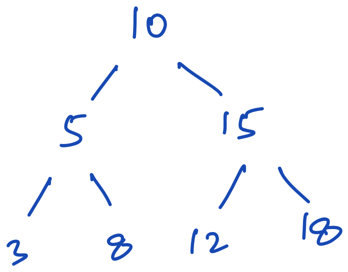
Nodes 6,7,8 are descendants of 3.

Subtree → Subtree of node x is the part of the tree which includes all the nodes that can be traveled from x.





Binary Tree → A tree in which all nodes have children <=2 (0,1,2)



```
class Node {
    int data;
    Node left, right;
    Node(x) {
        data = x
        left = right = NULL
    }
}
```

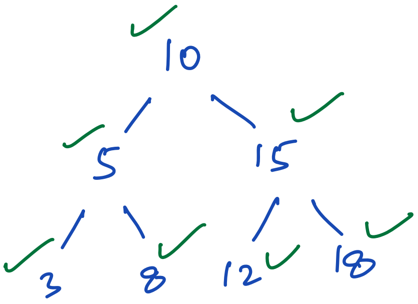
Tree traversal

1. Pre order traversal   Node left right
2. In order traversal   left Node right
3. Post order traversal   left right Node

1. Preorder Traversal

N L R

	Node	left			right		
output :	10	5	3	8	15	12	18
		N	left	right	N	left	right



```
void preorder (root) {
      if (root == null) return;
      print (root.data)          N
      preorder (root.left)       L
      preorder (root.right)      R
}
```

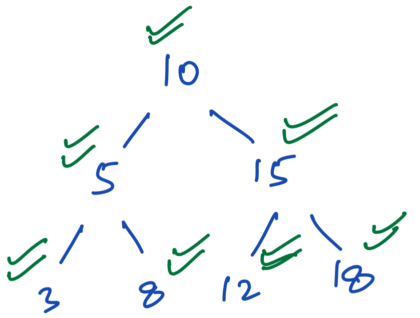
N → # of nodes

h → height of tree
= O(N)

TC = O(N)

SC = O(H)

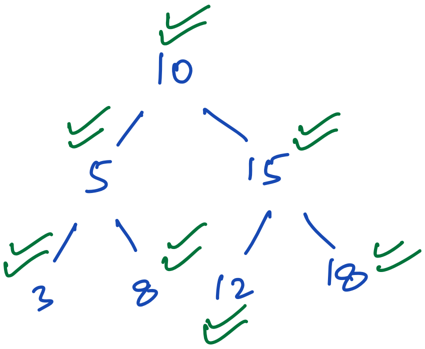
2. Inorder traversal   L N R



	left			node	right		
o/p :	3	5	8	10	12	15	18
	L	N	R		L	N	R

```
void inorder (root) {
        if(root == null)  return;
        inorder ( root.left)          L
        print ( root . data)          N
        inorder ( root . right )      R
    }
```

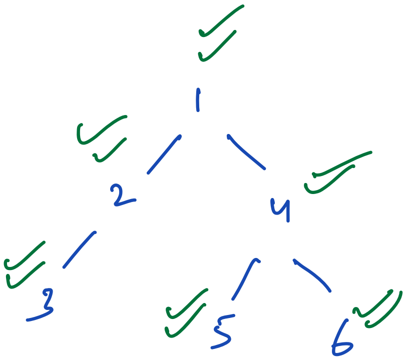
3. Post Order Traversal   L R N



	left			right			Node
o/p :	3	8	5	12	18	15	10
	L	R	N	L	R	N	

```
void postorder (root) {
        if(root == null)  return;
        postorder ( root.left)        L
        postorder ( root . right )    R
        print (root . data)           N
    }
```

Quiz



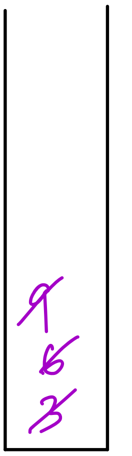
inorder : L N R

o/p : 3 2 1 5 4 6

Ques → Iterative inorder traversal

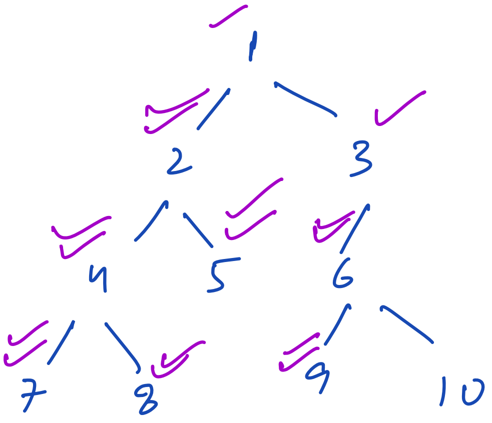
recursive ⟶ iterative
(use stack)

```
void inorder (root) {
        if (root == null) return;
        inorder (root.left)        L
        print (root.data)          N
        inorder (root.right)       R
}
```



stack

curr = ~~1~~ ~~2~~ ~~4~~ ~~7~~ null
~~7~~ null
~~4~~ ~~8~~ ~~null~~ ~~8~~ ~~null~~
~~2~~ ~~5~~ ~~null~~ . . . . .
~~1~~ ~~3~~ ~~6~~ ~~9~~ null
~~9~~ 6 . . . . .
. . . . .



o/p : 7 4 8 2 5 1 9 6 10

Code

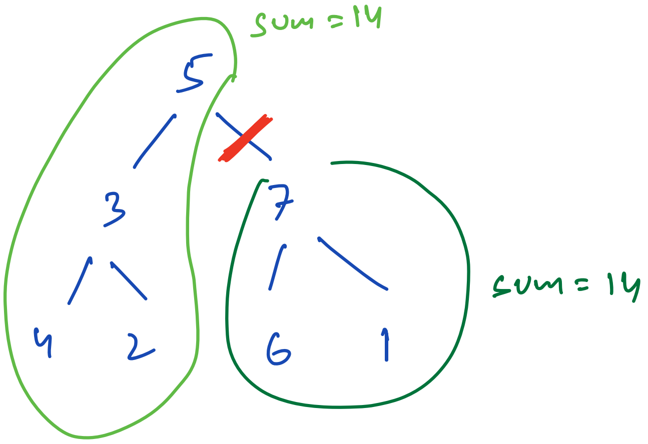
```
curr = root

while( curr != null || !st.isEmpty()) {

    if (curr != null) {
        st.push(curr)
        curr = curr.left        L
    }
    else {
        curr = st.pop()
        print(curr.data)        N
        curr = curr.right       R
    }
}
```

TC = O(N)

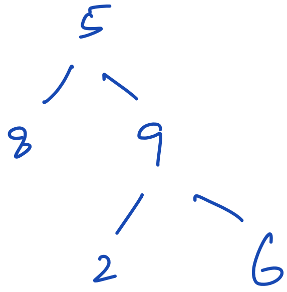
SC = O(n)

Question

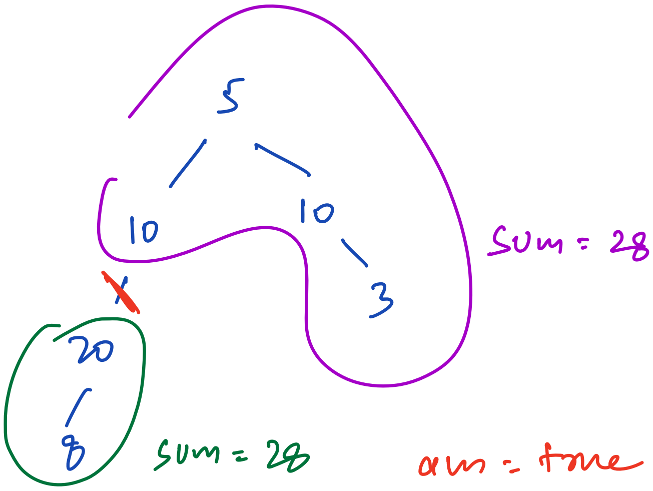
Given a root of a binary tree, return true if the tree can be split into 2 non-empty subtrees with equal sum, or false otherwise.



ans = true



ans = false



find a subtree sum = everything else's sum

= total sum − subtree sum

$$\boxed{\text{subtree sum} = \frac{\text{total sum}}{2}}$$

② post order

```
int sum( root) {
    if( root == null)   return 0;
    return root.data + sum( root.left) + sum( root.right );
}
```

① ②

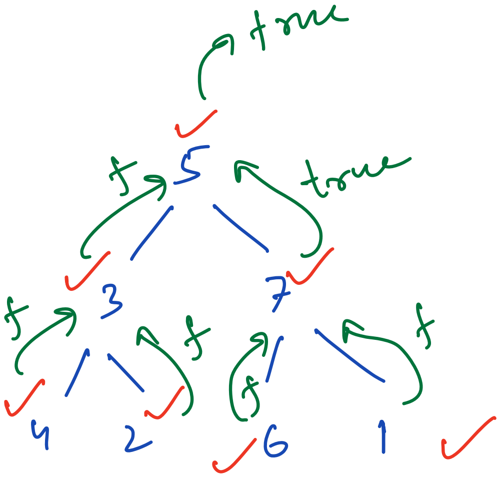
```
total Sum = sum( root)
if ( total Sum % 2  == 1)   return false
return   check( root, totalsum/2)
```

```
bool check( root , S ) {
    if( root == null)   return false
    if( check( root.left, S ) || check( root.right, S )) {
        return true
    }
    if ( sum( root)  == S)
```

return true

return false

3



totalSum = 28

$S = \frac{28}{2} = 14$

ans = true

TC = O(N)

SC = O(N)