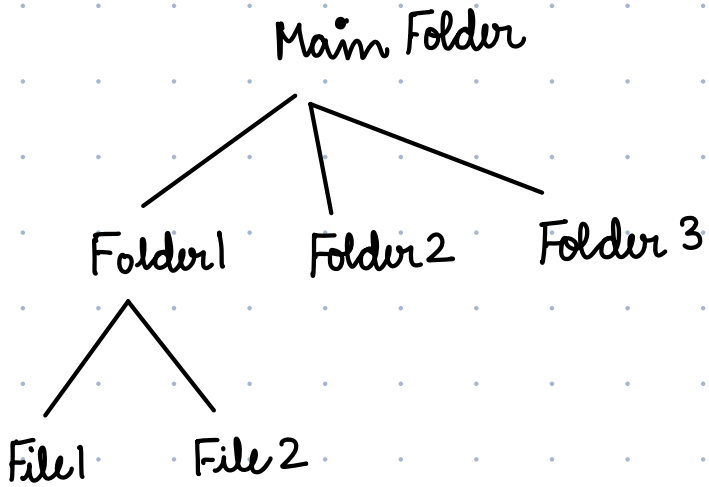
Arrays LL Stacks Queues - Linear

Trees - Non Linear

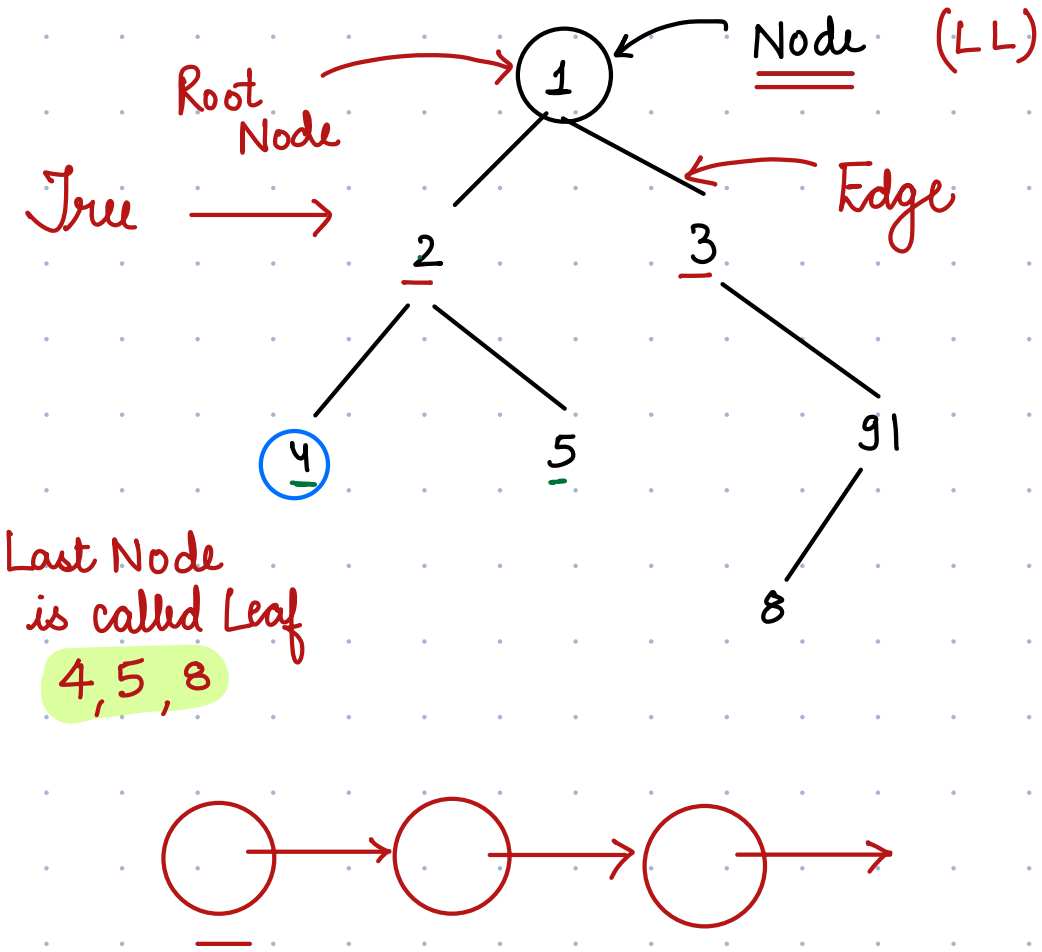
Family tree

Company hierarchy

File System



Hierarchy with the help of Number



First node of tree = Root

2 and 3 are called child of 1

1 is called parent of 2 and 3

Last Node is called Leaf

4, 5, 8

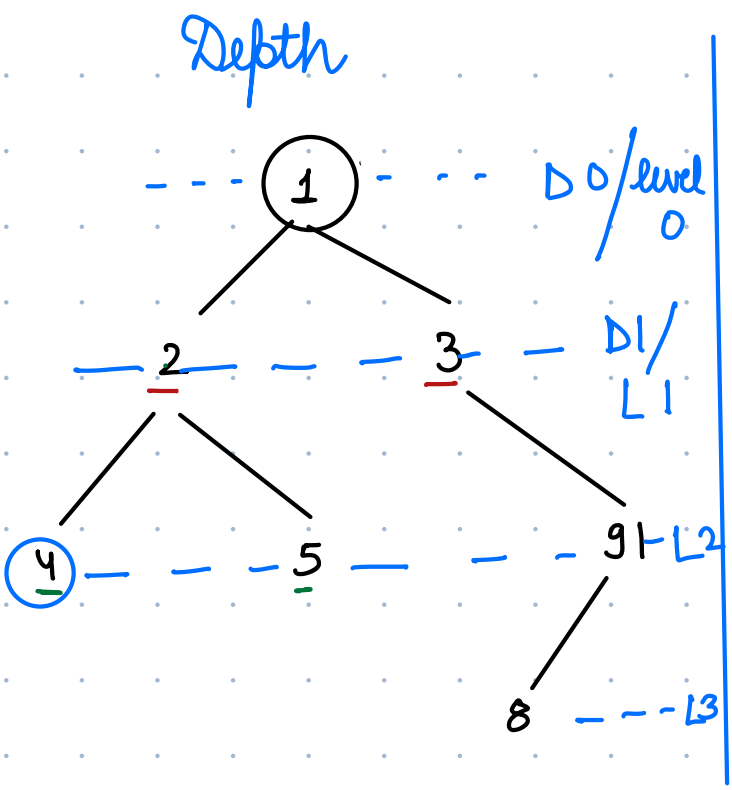
Subarray of an array
Substring of a string
subtree of a tree — part of tree with some nodes

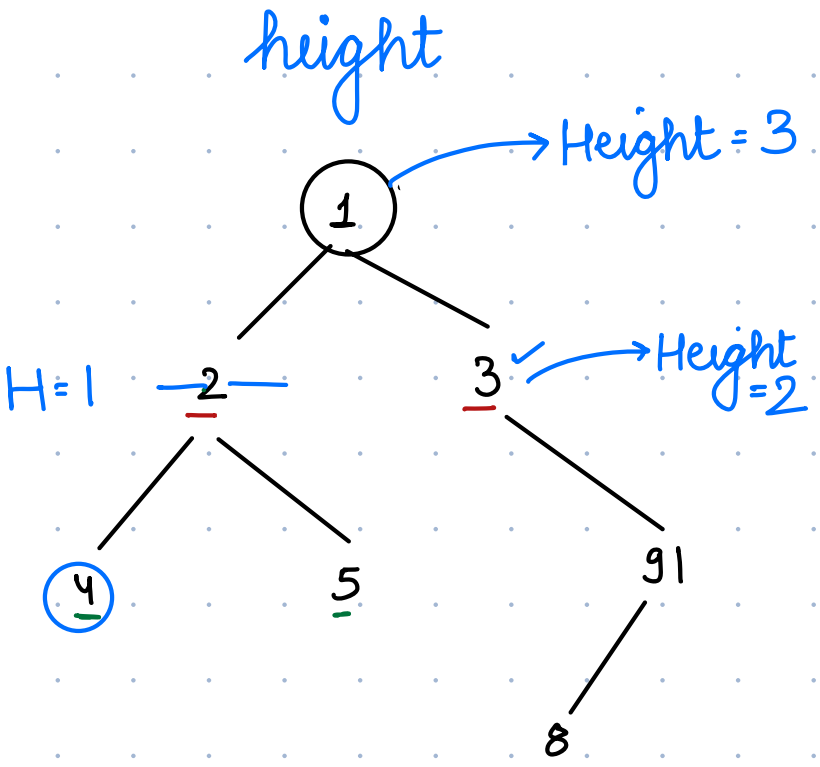
If 2 nodes have same parent, they are called siblings

2 & 3.

Ancestor — Root to Node path — All nods inside it.

Descendant — Node to leaf part — All nods inside it.





Depth is measured from root node

Height is measured from leaf node.

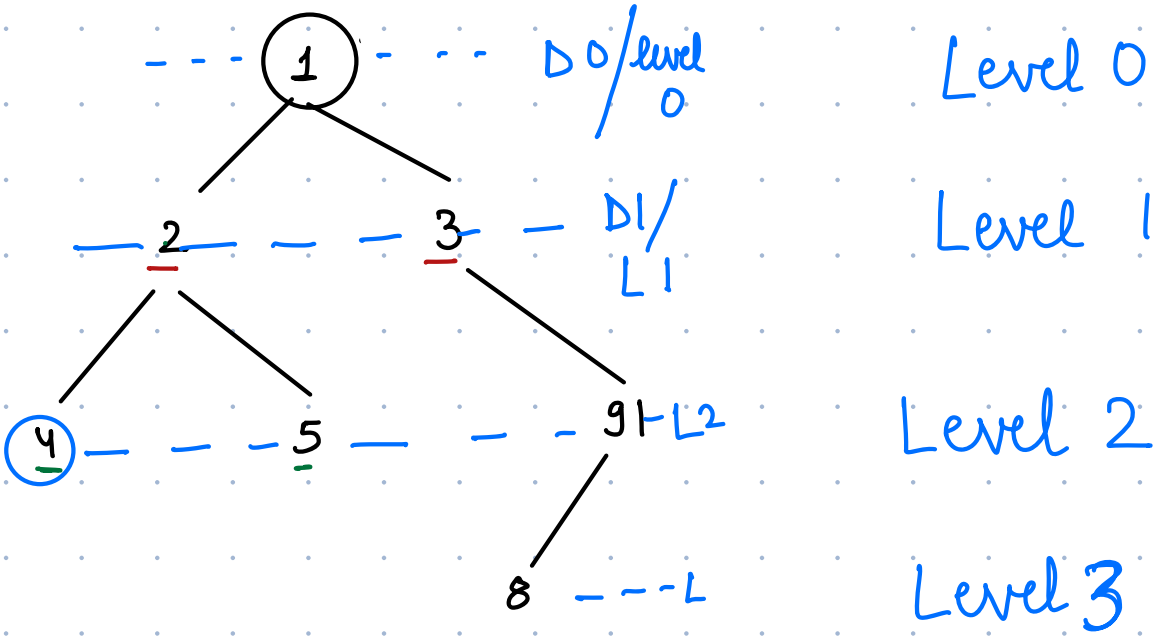
→ Can leaf Node be a subtree?

Yes

→ Do all nodes have a parent node?

No → Root node

- Depth = Level



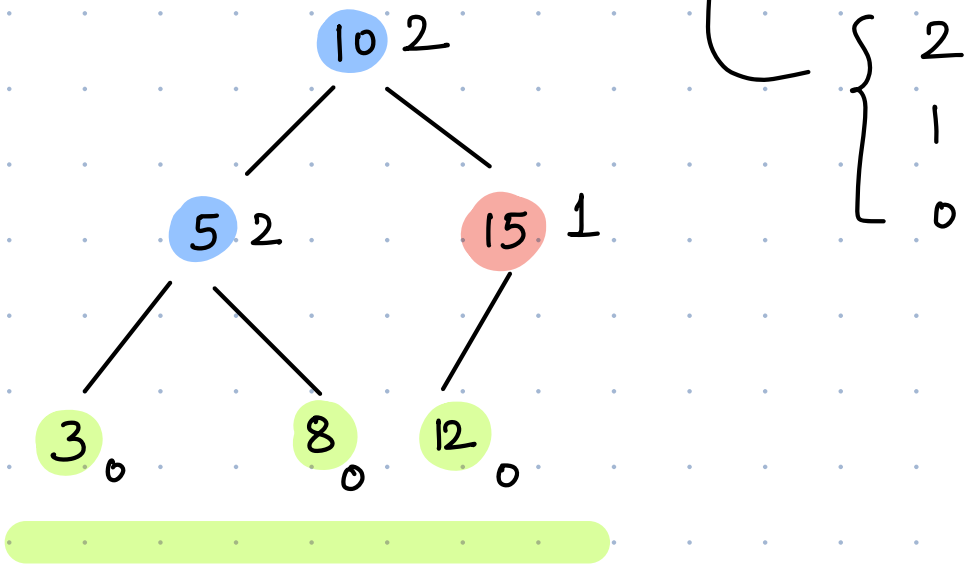
Q. - Height = Leaf

↳ Height of leaf node = 0

Can1 node be a tree

Yess!

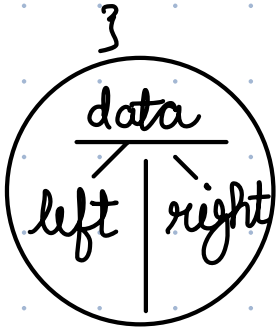
max. 2 children ← Binary Tree



---

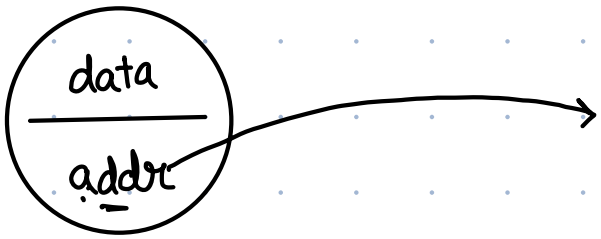
make node of a binary tree

```
class Node {
    int data
    Node left
    Node right
```



node of a LL

```
class Node
    int data
    Node next
```



Binary base 2 {0 1}

Decimal base 10 {0 — 9}

Octal base 8 {0 — 7}

Hexadecimal base 16 {0 — 9, A, B, C, D, E, F}

Example addresses —

E948

A41

12E

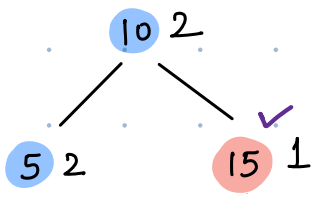
549

90A

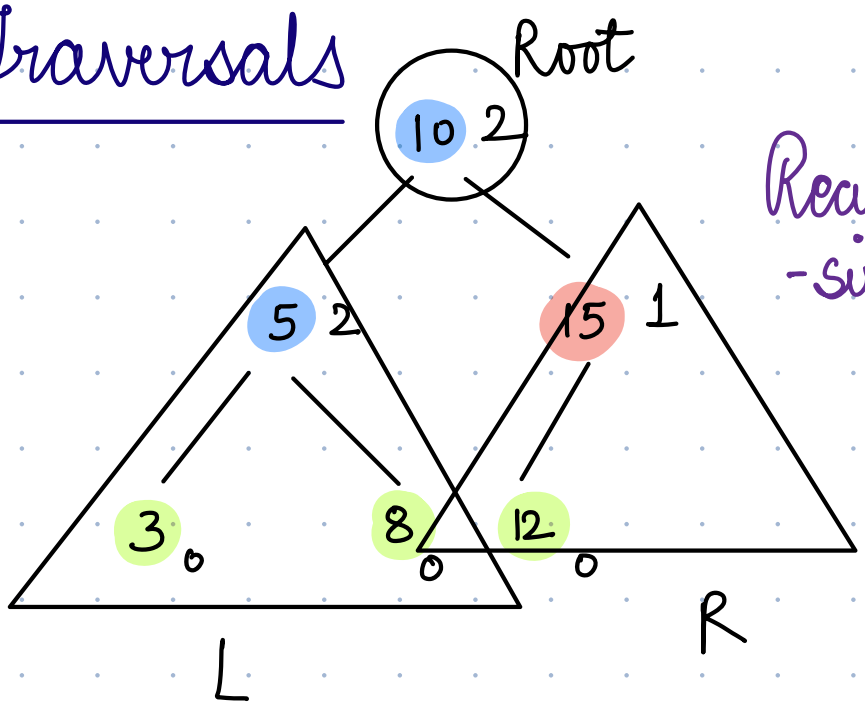
root = E948

A41

12E



Traversals



Recursive
- PreOrder ✓
- PostOrder ✓
- InOrder ✓

InOrder Iterative

L Root R

Sub - true

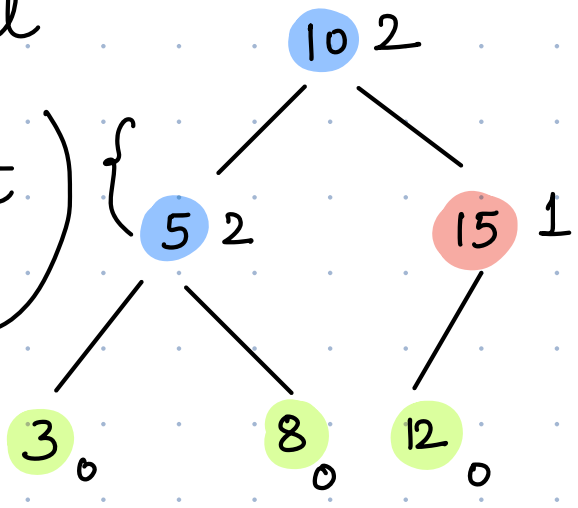
Sub - true

Left Root Right = InOrder

Root Left Right = PreOrder

Left Right Root = PostOrder

PreOrder Traversal




```
void inOrder (Node root) {
    inOrder(root.left)
    print (root.data);
    inOrder(root.right)
}
```

root

P = Print nodes of tree rooted at (10)

root.left

SP1 = Print nodes of tree rooted at (5)

Print (root.data)

root.right

SP2 = Print nodes of tree rooted at (15)

INORDER

root

P = Print nodes of tree rooted at (10)

SP1 = Print nodes of tree rooted at (5) root.left

SP2 = Print nodes of tree rooted at (15) root.right

Print (root.data)

POST ORDER

root

P = Print nodes of tree rooted at (10)

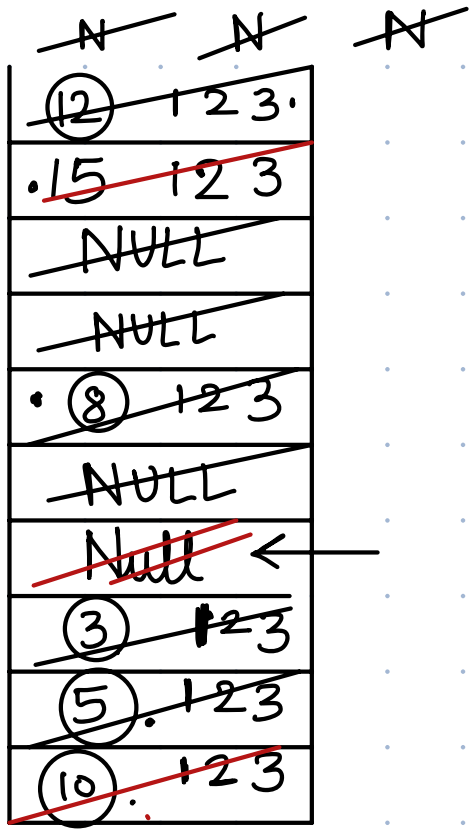
Print (root.data)

SP1 = Print nodes of tree rooted at (5) root.left

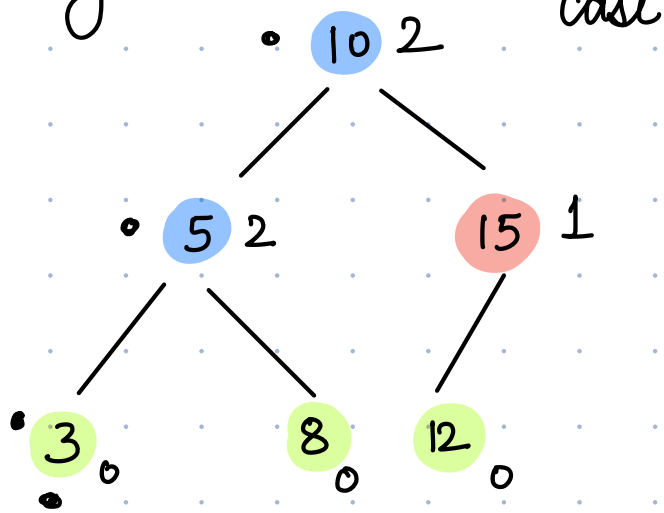
SP2 = Print nodes of tree rooted at (15) root.right

PRE ORDER

```
void inOrder(Node root)
    if (root == null) { return; }
    inOrder(root.left)
    print(root.data);
    inOrder(root.right)
}
```



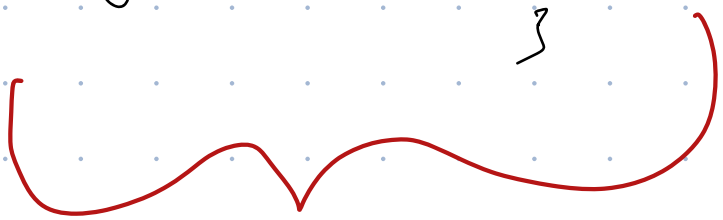
Dry Run → Base case



3 5 8 10 12 15

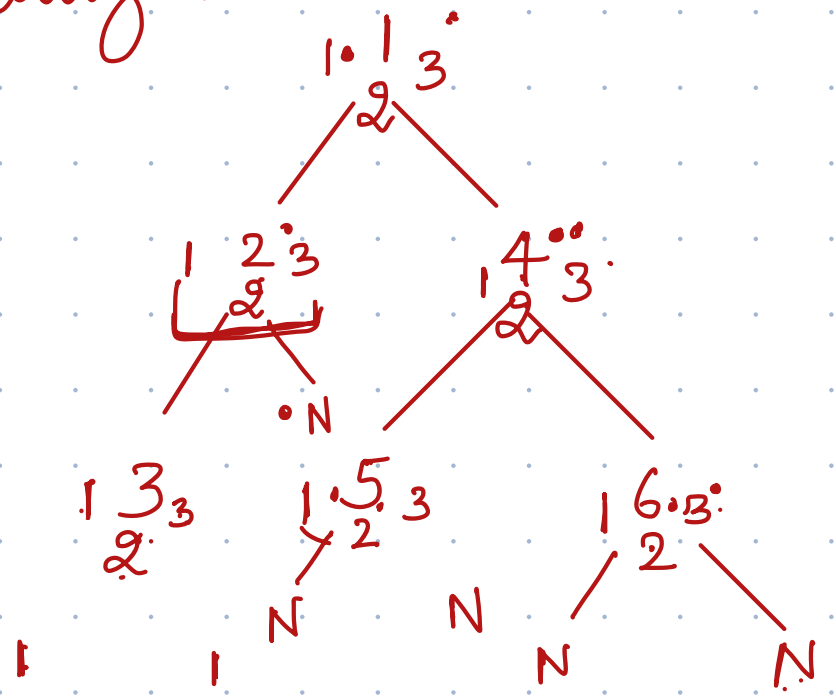
```
void preOrder(Node root)
    if (root == null) { return; }
    print(root.data);
    preOrder(root.left)
    preOrder(root.right)
}
```

```
void postOrder(Node root)
    if (root == null) { return; }
    postOrder(root.left)
    postOrder(root.right)
    print(root.data);
}
```



Dry Run = Exercise

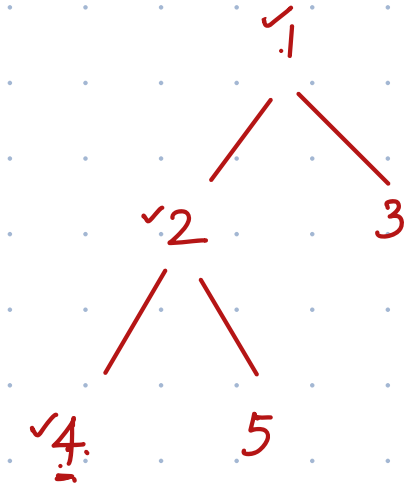
Quiz:



Inorder

3 2 1 5 4 6

Iterative InOrder :



4
2
5
1
3

Start with !

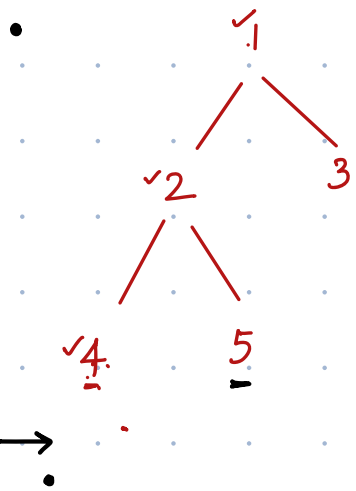
Push Element → Push its left

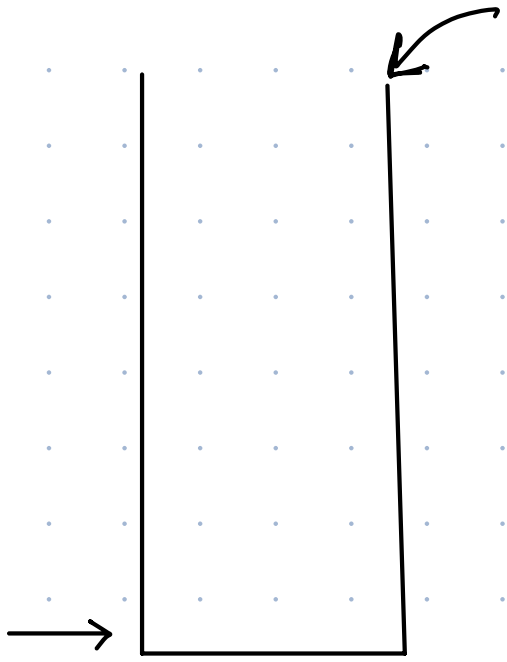
Pop Element, Print element → Push its right

```
curr = root
while (curr != NULL || st.isEmpty() == false) {
    if (curr != NULL) {
        st.push(curr)
        curr = curr.left
    } else {
        curr = st.pop()
        print(curr.data)
        curr = curr.right
    }
}
```

Dry Run :

```
curr = root
while (curr != NULL || st.isEmpty() == false) {
    if (curr != NULL) {
        st.push(curr)
        curr = curr.left
    } else {
        curr = st.pop()
        print(curr.data)
        curr = curr.right
    }
}
```





curr = ~~4~~ ~~NULL~~ ~~2~~ ~~5~~ NULL

~~5~~ NULL 1 → ~~3~~ NULL

~~3~~

NULL

4 2 5 1 3

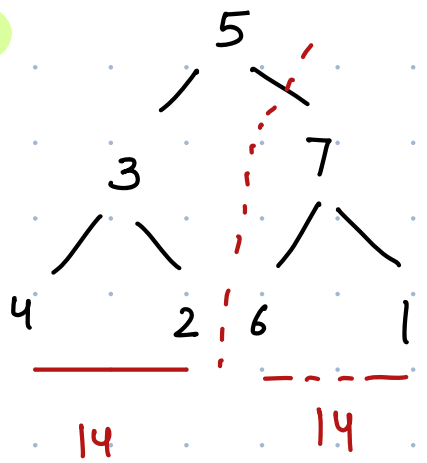
Equal Tree Partition :

Recursion

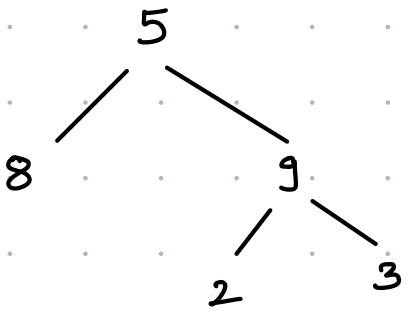
Given
Root of binary tree

Return true if the tree can be split into 2 non-empty subtrees of equal sums

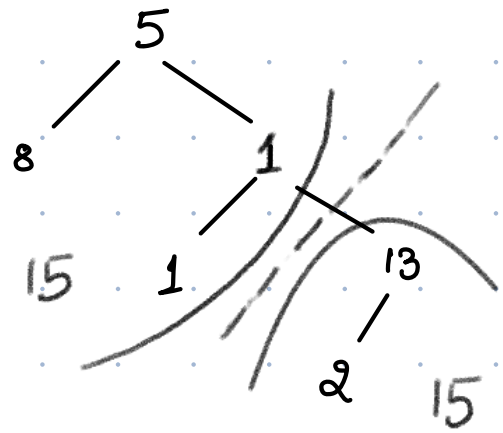
Otherwise return false.



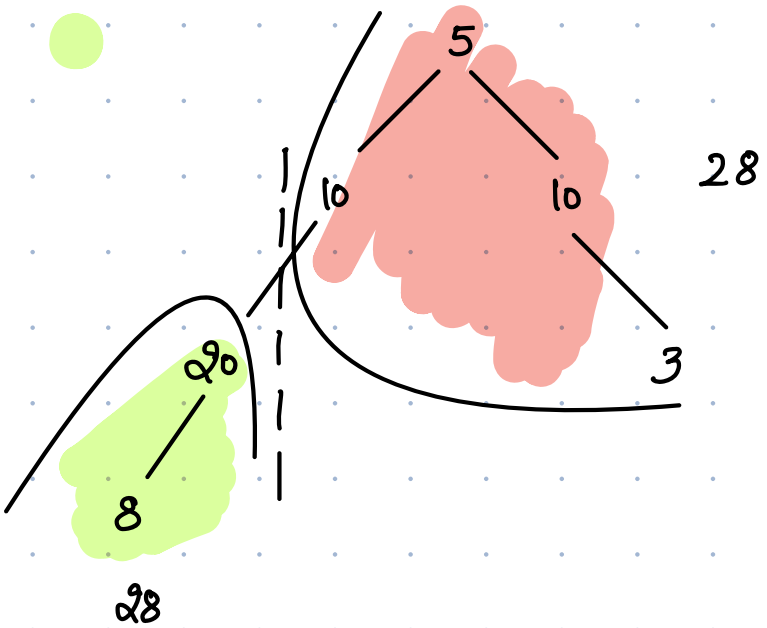
True



- False



True



Yes

```
function sum ( Node root ) {
  if (root == null) { return 0; }
  int s1 = sum( root.left);
  int s2 = sum ( root.right);
  return s1 + s2 + root.data;
}
```

boolean

function hasSubtreeWithHalfSum (Node root, int totalSum) {

if (root == null) { return false }

base case for odd

SL = sum (root.left)

if (SL == totalSum / 2) { return true }

SR = sum (root.right)

if (SR == totalSum / 2) { return true }

ans = hasSubtreeWithHalfSum (root.left, totalSum)

if (ans == true) { return true }

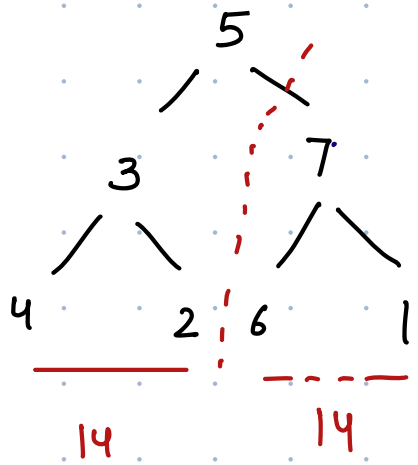
ans = hasSubtreeWithHalfSum (root.right, totalSum)

if (ans == true) { return true }

return false;

}

28



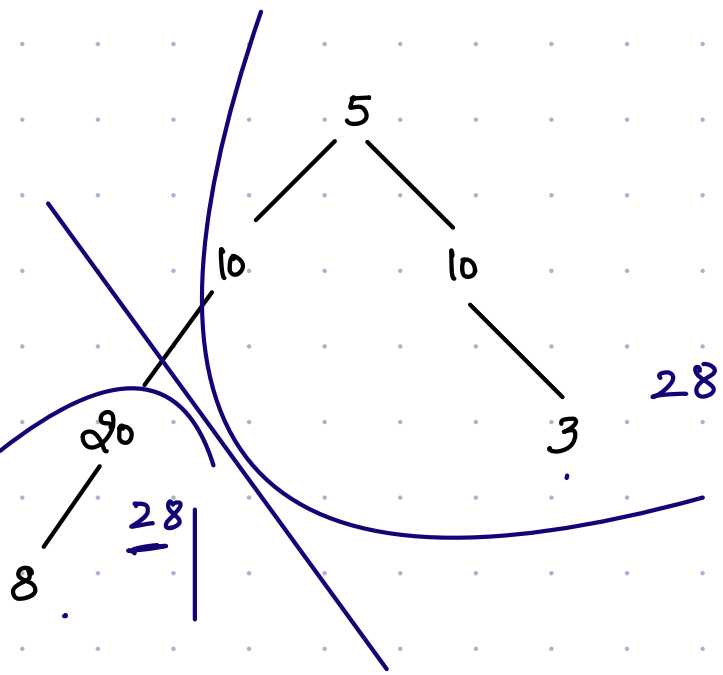
5

(3) = 9

(7) = 14 = 28/2

Rs

TS = 56



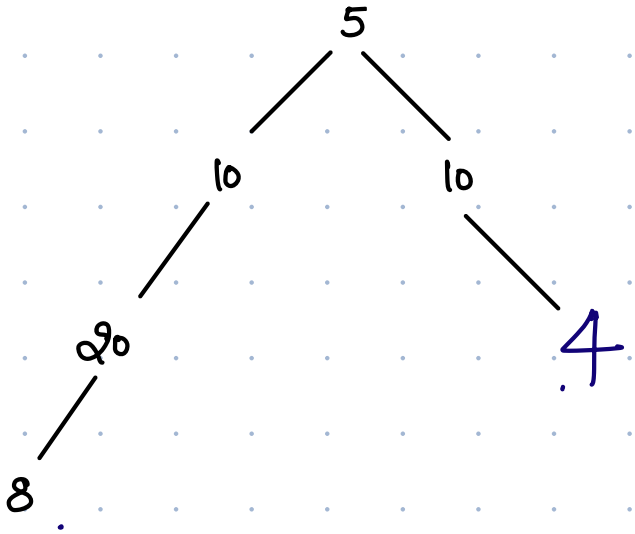
(5)

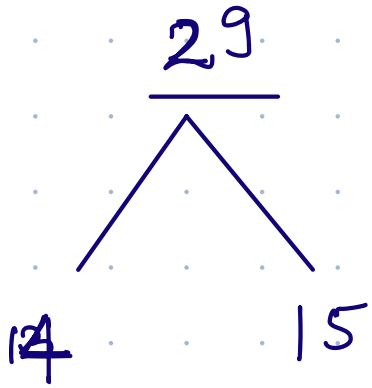
(10) = 38 ✗

(10) = 13 ✗

(10)

(20) = 28 true


5
10
10
20
4
8


29
14
15