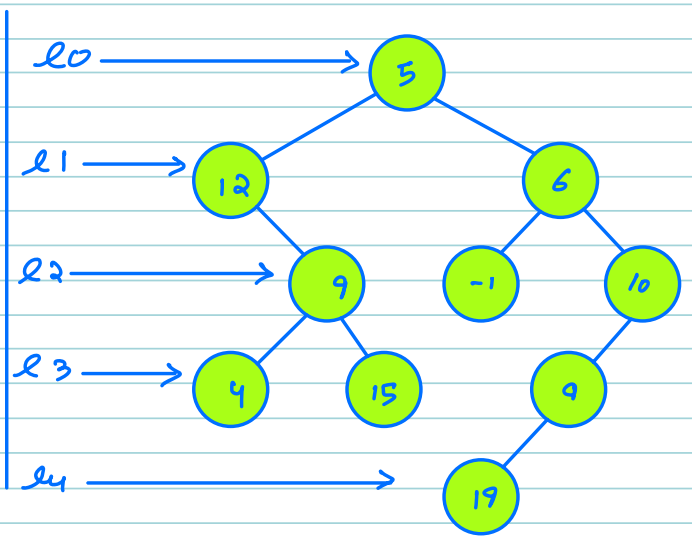
Trees 2 : Views & Types

TODAY'S CONTENT

- Level order Traversal
- Left view & right view
- Vertical Level order
- Top view & Bottom view
- Types of Binary Tree
- check Height Balanced Tree

Question :- Level order Traversal

output :- 5, 12, 6, 9, -1
10, 4, 15, 9, 19



quiz 1 :- Will last level node always be a leaf node?
↳ yes

quiz 2 :- which traversal is best to print the nodes from top to bottom ↳ Level order traversal.

level = ~~x~~ x ~~x~~ 1

Queues.

5 12 6 9 -1 10 4 15 9 19

output = 5, 12, 6, 9, -1, 10, 4, 15, 9, 19

PSEUDO CODE

```
void LevelOrderTraversal(Node root)
    Queue<Node> q;
    q.add(root);
    while(q.size()>0){
        Node front = q.peek()
        q.remove()
        print(front.data);
        if(front.Left != null){
            q.add(front.Left);
        }
        if(front.Right != null){
            q.add(front.right);
        }
    }
}
```

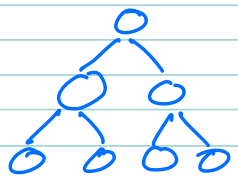
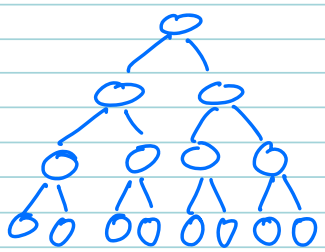
TC → O(N)

Space complexity Analysis

⊙ At one time how many different level can be there in queue.

↳ At max 2

Now, let see at max how Nodes can be there in 2 levels.

		n	last level	
Ex1		3	2	
Ex2		7	4	last level $n/2$
Ex3		15	8	

Ulty second last level $\frac{n}{4}$

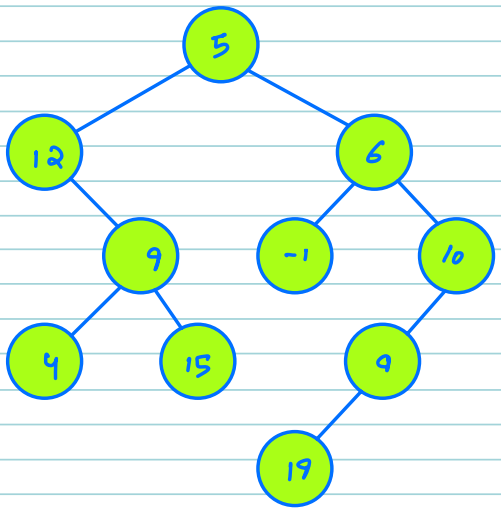
↳ So, the last level have $\left(\frac{n+1}{2}\right)$ nodes in case of complete tree

$$SC \longrightarrow \frac{n}{2} + \frac{n}{4}$$

$$\frac{3n}{4}$$

$$SC \leq O(N)$$

Followup :- Level order traversal



output :-

5 ℓ_n

12 6 ℓ_n

9 -1 10 ℓ_n

4 15 9 ℓ_n

19

DRY RUN

N = 1 0 2 1 0 3 2 1 0 3 2 1 0 0

5 12 6 9 -1 10 4 15 9 19

output

5

12 6

9 -1 10

4 15 9

19

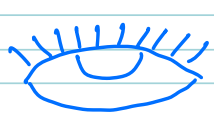
PSEUDO CODE

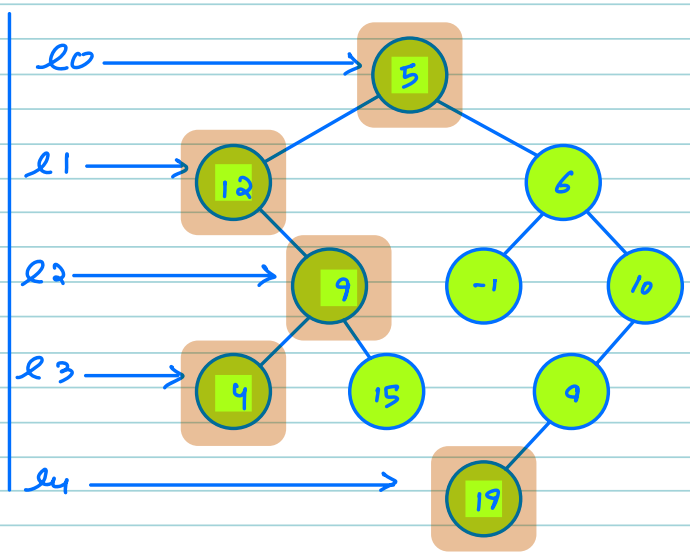
```
void levelOrderTraversal ( Node root ){
    Queue < Node > q ;
    q . add ( root );
    while ( q . size ( ) > 0 ){
        int N = q . size ( );
        for ( i = 1 ; i <= N ; i++ ){
            Node front = q . peek ( )
            q . remove ( )
            print ( front . data );
            if ( front . left != null ){
                q . add ( front . left );
            }
            if ( front . Right != null ){
                q . add ( front . right );
            }
        }
        Print (ln) ;
    }
}
```

TC — O(N)

SC — O(N)

Question :- Left view of a tree.





output :- 5, 12, 9, 4, 19

IDEA :- First Node of each level.

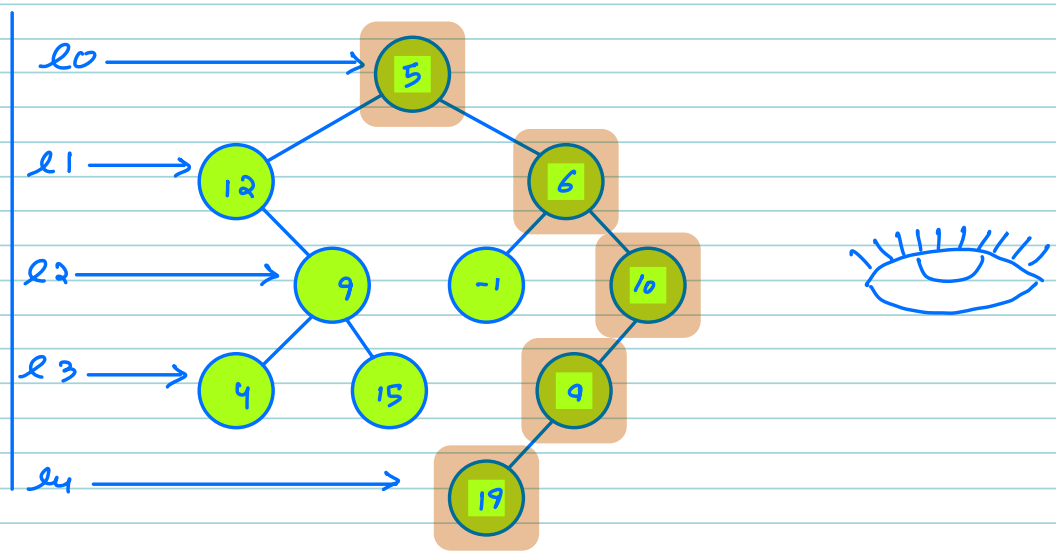
PSEUDO CODE

```
void levelOrderTraversal ( Node root ){
    Queue < Node > q ;
    q . add ( root );
    while ( q . size ( ) > 0 ){
        int N = q . size ( );
        for ( i=1 ; i<=N ; i++ ){
            Node front = q . peek ( )
            q . remove ( );
            if ( i == 1 ){ Print ( front . data ); }
```

```
if ( front.left != null){
    q.add( front.left);
}
if ( front.right != null){
    q.add (front.right);
}
}
}
}
```

TC — O(N)
SC — O(N)

Question :- Right view of a tree



output :- 5, 6, 10, 9, 19

IDEA :- Last Node of each level.

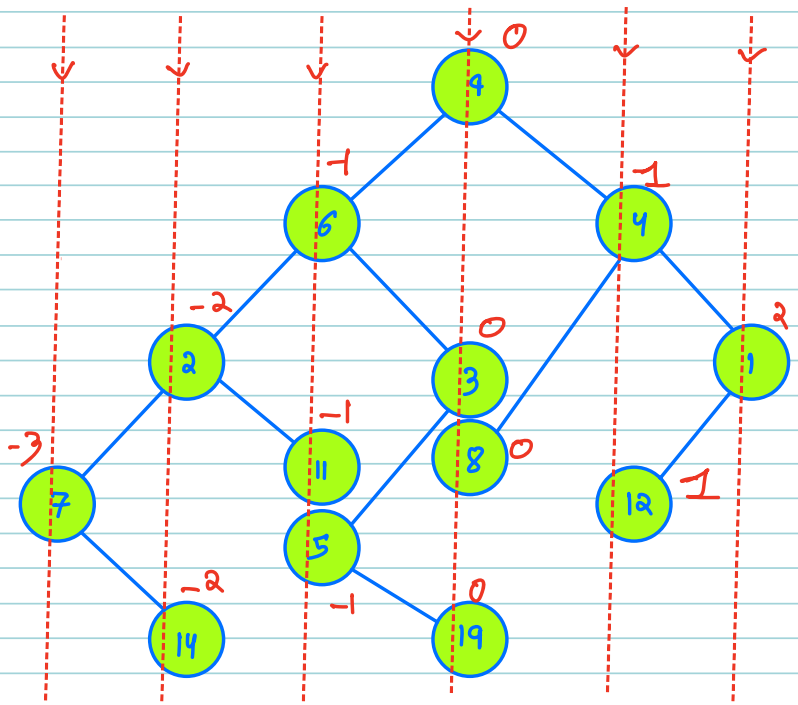
PSEUDO CODE

```
void levelOrderTraversal ( Node root ){
    Queue <Node> q ;
    q . add ( root );
    while ( q . size ( ) > 0 ){
        int N = q . size ( );
        for ( i=1 ; i<=N ; i++ ){
            Node front = q . peek ( )
            q . remove ( );
            if ( i == N ){ Print ( front . data ); }
            if ( front . left != null ){
                q . add ( front . left );
            }
            if ( front . Right != null ){
                q . add ( front . right );
            }
        }
    }
}
```

TC — O(N)

SC — O(N)

Question :- Vertical level order traversal (L → R)



Expected O/P

→ 7

→ 2, 14

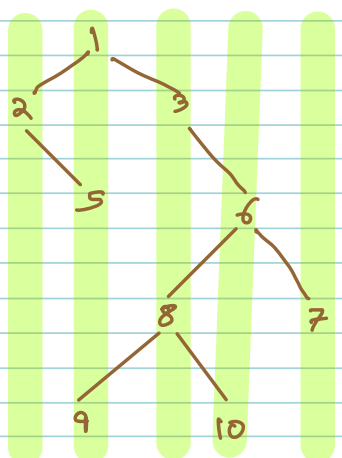
→ 6, 11, 5

→ 9, 3, 8, 19

→ 4, 12

→ 1

Quiz :- Consider the following binary Tree.



output :- 2 1 5 9 3 8 6 10 7

Q How will identify the element of Same Vertical Line.

IDEA :- Same Distance on Left & Right from root

↳ Let, Consider the Vertical level of root as 0.

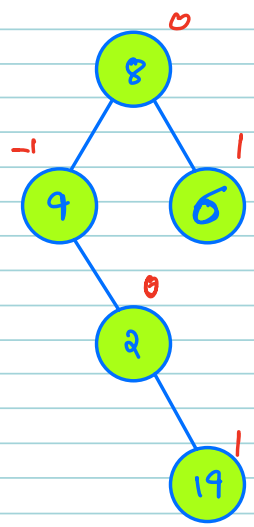
↳ So, when going Left do Vertical level of root - 1

And, when going right do Vertical level of root + 1

Approach :-

↳ Do traversal, Pass the level

↳ On each Node, store the data corresponding to each level in map



(9, 2, 19, 8, 6)

Expected o/p

-1 → 9

0 → 8 2

1 → 6 19

↑ Key of map

Inorder

-1 → 9

0 → 2 8

1 → 19 6

Map Values are out of order.

Pre-order

(8, 9, 2, 19, 6)

-1 → 9

0 → 8 2

1 → 19 6

Map values are out of order

Post-order

(19, 2, 9, 6, 8)

-1 → 9

0 → 2 8

1 → 19 6

out of order.

Hence, we can say Reursive traversal will not work because left is getting Processed before Right. So 6 can never be before 19.

Hashmap

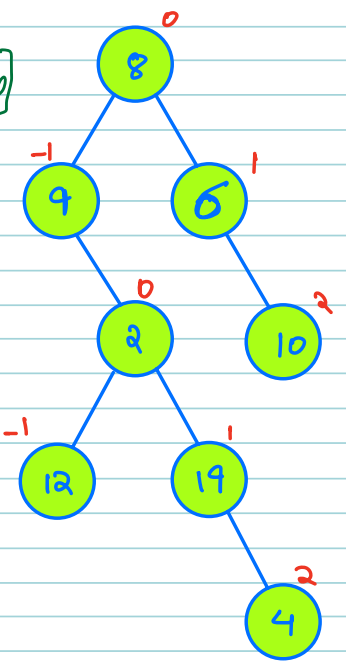
Key [Vertical level] | Value [Nodes in that level]

-1 → 9, 12

0 → 8, 2

1 → 6, 19

2 → 10, 4



Hashmap

Key [Vertical level] | Value [Nodes in that level]

0 → 8, 2

-1 → 9, 12

2 → 10, 4

1 → 6, 19

Node Level

Queue

~~(8,0)~~ ~~(9,-1)~~ ~~(6,1)~~ ~~(2,0)~~ ~~(10,2)~~ ~~(12,1)~~ ~~(19,1)~~ ~~(4,2)~~

PSEUDO CODE

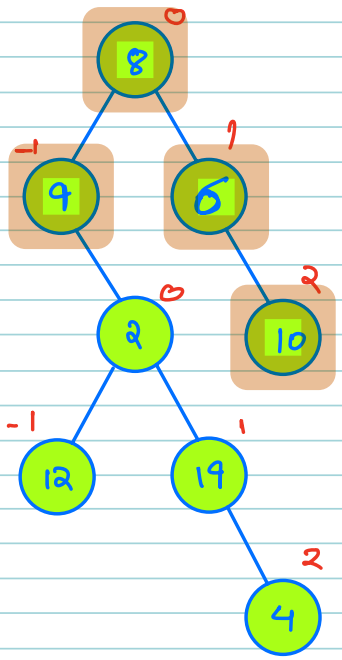
```
class Pair {
    Node first;
    int second;

    public pair (x, y) {
        first → x
        second → y
    }
}
```

Key (vertical levels) → Values [List of Node data]

```
HashMap < int , List < int >> hmap;
Queue < Pair > q;
maxL = -∞ , minL = ∞
q. insert (new Pair (root, 0));
while (q. size () > 0) {
    Pair f = q. peek();
    q. remove();
    Node t = f. first();
    int l = f. second();
    maxL = max(maxL, l); minL = min(minL, l);
    hm [l]. add ( t. val);
    if ( t. left != null) {
        q. add (new pair(t. left, l-1));
    }
    if ( t. right != null) {
        q. add ( new pair (t. right, l+1));
    }
}
for ( i = minL ; i <= maxL ; i++){
    Print ( hm. get (i));
}
```

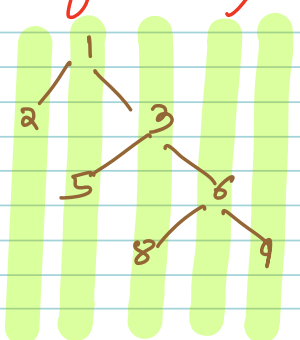
TC → O(N)
SC → O(N)

Followup :- Top View



output → 9, 8, 6, 10

Ques :- Consider the following binary Tree.



output :- 2, 1, 3, 6, 9

IDEA :- Corresponding to each key in map there is a list. We just need to print the first value of list.

PSEUDO CODE

```
class Pair {
    Node first;
    int second;

    public pair (x, y) {
        first → x
        second → y
    }
}
```

Key (vertical levels) → Values [List of Node data]

```
Hashmap < int , List < int >> hmap;
Queue < Pair > q;
maxL = -∞ , minL = ∞
q. insert (new Pair (root, 0));
while (q. isEmpty () > 0) {
    Pair f = q. peek();
    q. remove();
    Node t = f. first();
    int l = f. second();
    maxL = max (maxL, l); minL = min (minL, l)
    hm [l]. add ( t. val);
    if ( t. left != null) {
        q. add (new pair (t. left, l-1));
    }
    if ( t. right != null) {
        q. add ( new pair (t. right, l+1));
```

T.C → O(N)

SC → O(N)

```
    }
}

for ( i = minL ; i <= maxL ; i++){
    print ( hm[i].get(0) );
}
```

Follow up :- Bottom view.

IDEA :- Last element in each Vertical level.

Break :- 10:40 <-> 10:50

TYPES OF BINARY TREE (B.T.)

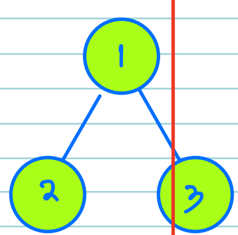
Question :- Check height balanced Tree

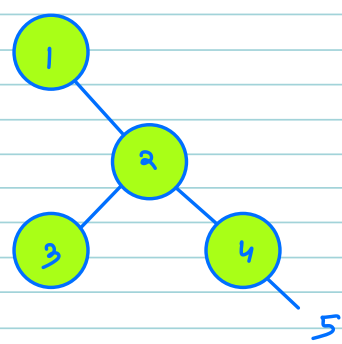
↳ A tree in which ∀ nodes

$|\text{LST height} - \text{RST height}| <= 1$

Note :- height in terms of Nodes.

Ex





TC > O( )

SC → O(N)

```
    }
}

for ( i = minL ; i <= maxL ; i++){
    Print ( hm[i].get(0) );
}
```

Follow up :- Bottom view.

IDEA :- Last element in each Vertical level.

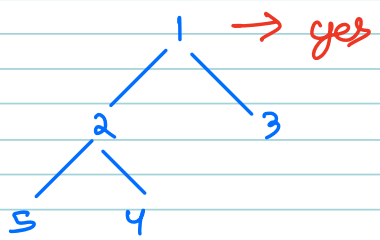
Break :- 10:40 <-> 10:50

Types Of Binary Tree (B.T.)

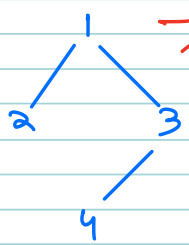
(1) Proper BT

↳ ∀ nodes 0 or 2 children

Ex



→ yes

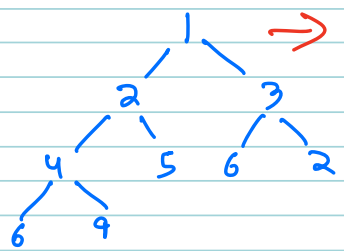


→ No

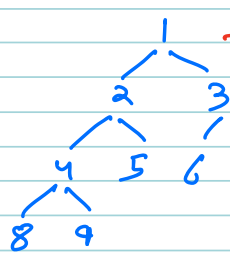
(2) Complete BT

↳ All levels must be completely filled except possibly the last level, which is filled left to right

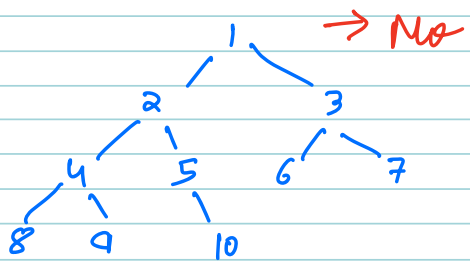
Ex



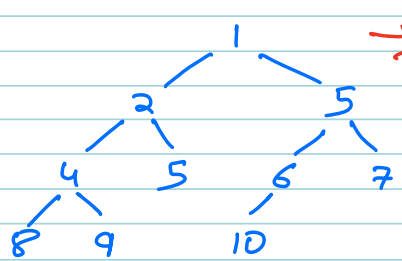
→ yes



→ No



→ No

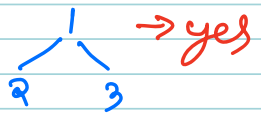


→ No

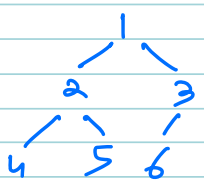
③ **Perfect BT**

↳ All levels are completely filled

Ex



→ yes



→ No.

Quiz 6 :- Perfect Binary Trees are also :-

↳ Both [Proper & Complete]

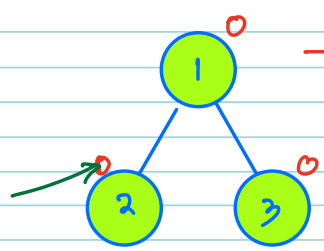
Question :- Check height balanced Tree

↳ A tree in which ∀ nodes

$|\text{LST height} - \text{RST height}| <= 1$

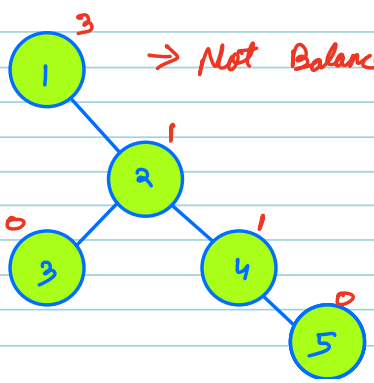
Note :- height in terms of Nodes.

Ex

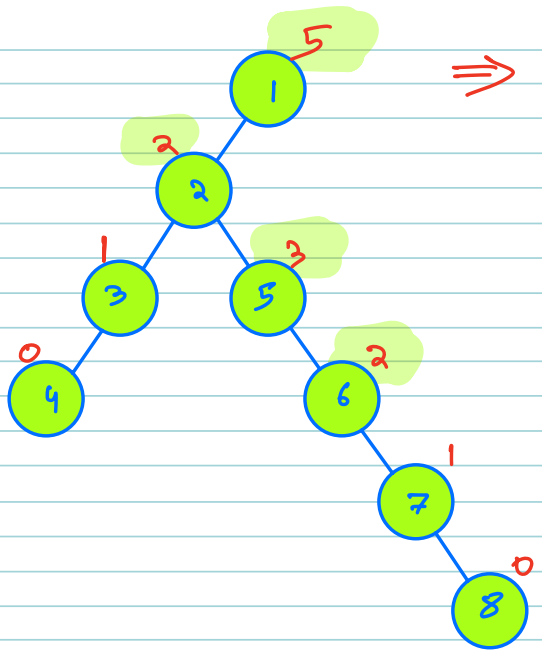


→ Balanced

Diff



→ Not Balanced

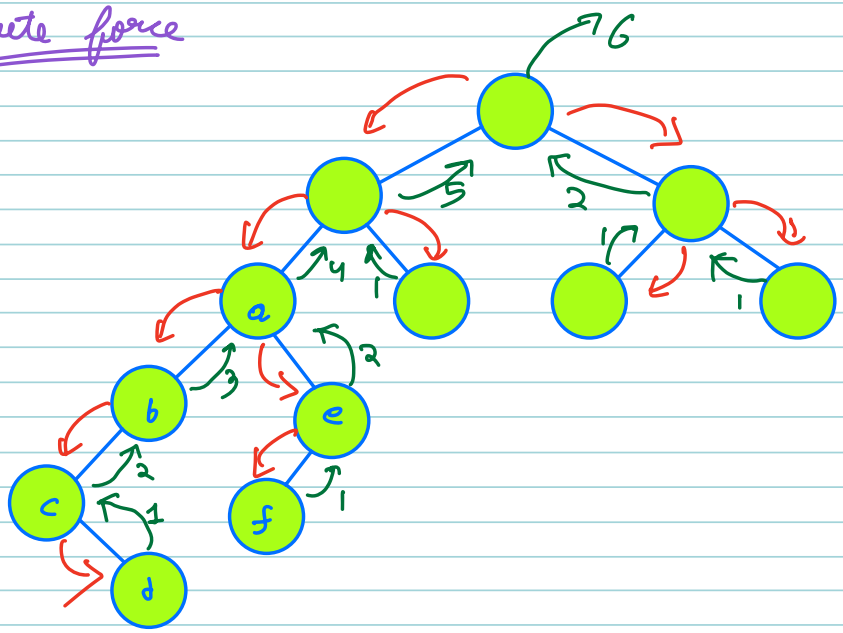


⇒ Not Balanced

Q How you will find height of tree ?

↳ Max(LST height, RST height) + 1

Brute force



Quiz 7 :- which traversal is best to use when finding the height of tree?

↳ Post order

```
int height (Node root) {
    if (root == null) return 0;
    int LH = height (root.Left);
    int RH = height (root.right);
    return Max(LH, RH) + 1;
}
```

Post order.

PSEUDO CODE

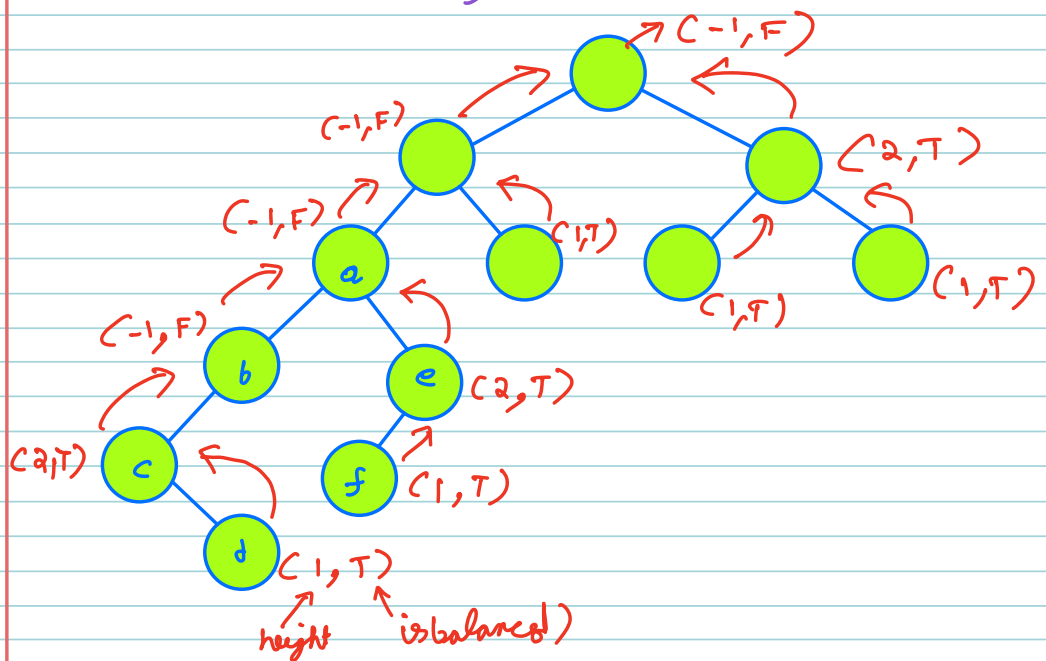
```
boolean isHb;

boolean isBalanced (Node root) {
    isHb = True;
    height (root);
    return isHb;
}
int height (Node root) {
    if (root == null) return 0;
    int LH = height (root.Left);
    int RH = height (root.right);
    if (Abs (LH - RH) > 1) { isHb = False; }
    return Max(LH, RH) + 1;
}
```

TC - O(N)

SC - O(H)

⇒ OPTIMIZATION

↳ We have to think how we can solve without using global variable.



IDEA :- Each Node will return object this class.

```
class Pair {
    int height;
    boolean isBalanced;
}
```

* At each Node we will check.
  - → If Left Side Balance
  - → Is Right Side Balance
  - → Am I balanced.

↳ If yes, then return pair of (height, True) else, (-1, False);

PSEUDO CODE

```
boolean isBalanced(Node root){
    Pair P = helper(root)
    return P.isBalanced;
}

Private Pair helper(Node root){
    if(root == null) return new pair(0, True);

    Pair LP = helper(root.Left);
    Pair RP = helper(root.right);

    if(LP.isBalanced == False || RP.isBalanced
                                 == False){
        return new pair(-1, False);
    }else if(Abs(LP.height - RP.height) > 1){
        return new pair(-1, False);
    }else{
        return new pair(Max(LP.height, RP.height)
                        + 1, True);
    }
}
```

TC → O(N)

SC → O(H)