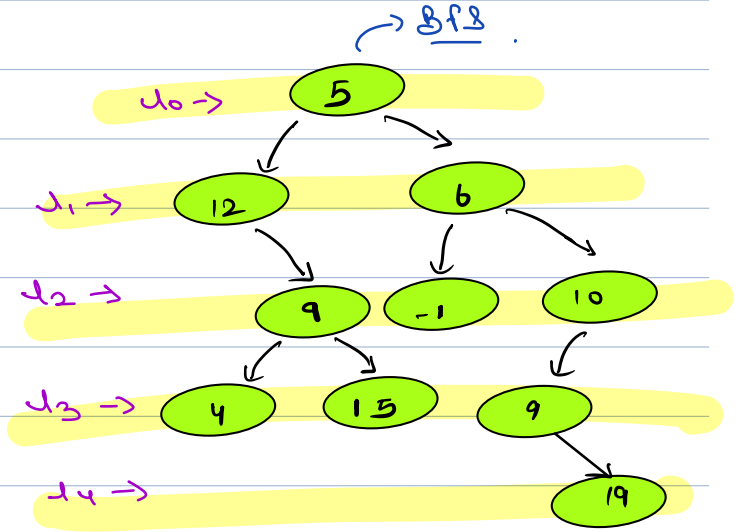
Today's content →

Level order Traversal
Left view & right view
Vertical Level Order
Top view & Bottom view
Types of B.T.
check Height Balanced

# 8) Level order Traversal

O/P :- 5 12 6 9 -1
10 4 15 9 19




5 12 6 9 -1 10 4 15

| n | last level |
|---|---|
| 3 | 2 |
| 7 | 4 |
| 15 | 8 |

last level $\frac{n}{2}$ , second last level $\frac{n}{4}$

Last level at best can have, $\frac{(n+1)}{2}$

In the last level approx $\frac{n}{2}$ elements are there.

```
void levelOrder(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);
        }
    }
}
```

T.C → O(n)

max space → $\frac{n}{2} + \frac{n}{4}$

S.C → O(n).

# * Level Order Traversal - 2 :-

**Output**

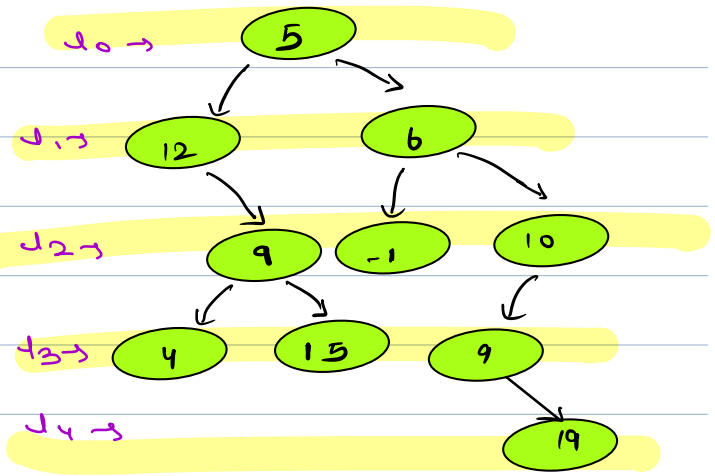
5 \n

12 6 \n

9 -1 10 \n

4 15 9 \n

19




n = 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0

~~5~~ ~~12~~ ~~6~~ ~~9~~ ~~-1~~ ~~10~~ ~~4~~ ~~15~~ ~~9~~ ~~19~~

5 \n

12 6 \n

9 -1 10 \n

4 15 9 \n

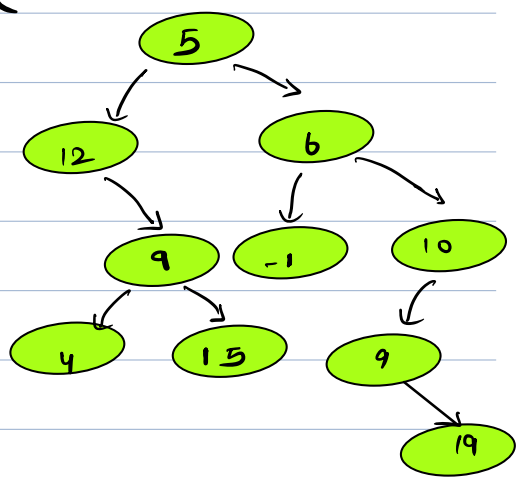
19

5 \n

12 6 \n

~~5~~ ~~12~~ ~~6~~ 9 -1 10

```
void levelOrder(Node root){
    Queue<Node> q;
    q.add(root);
    while(q.size()>0){
        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("\n");
    }
}
```

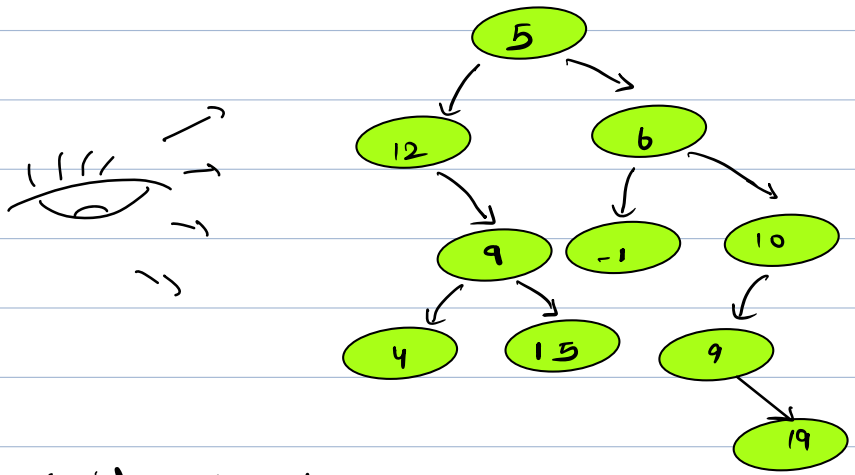



T.C → O(n)

S.C → O(n)

# Left View :-

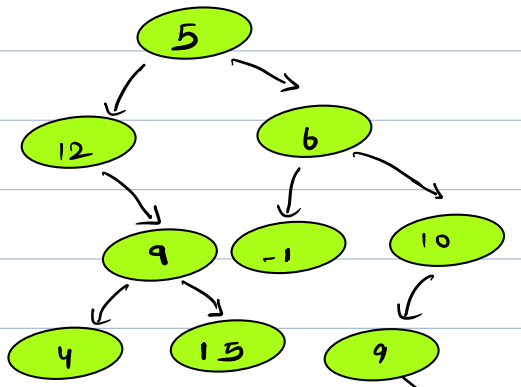
o/p → 5 12 9
4 19




```
void levelOrder(Node root){
    Queue<Node> q;
    q.add(root);
    while(q.size()>0){
        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("\n");
    }
}
```

if (i == 1)

# Right View :-



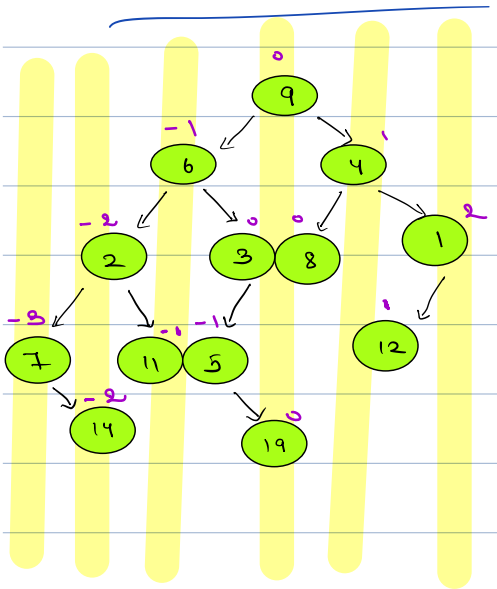

O/P

5 6 10 9 19

```
void levelOrder(Node root){
    Queue<Node> q;
    q.add(root);
    while(q.size()>0){
        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("\n");
    }
}
```

if(i==n)

# Vertical level order traversal (L-R)



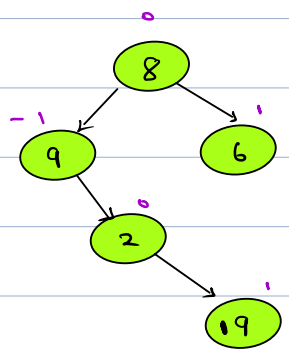

Expected O/P :-

→ 7
→ 2 14
→ 6 11 5
→ 9 3 8 19
→ 4 12
→ 1

Top View

7 2 6 9 4 1

Example :-




Expected O/P

-1 → 9
0 → 8, 2
1 → 6, 19

InOrder

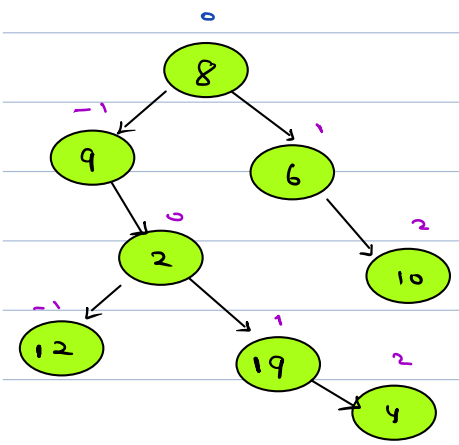
-1 → 9
0 → 2, 8
1 → 19, 6

PreOrder

-1 → 9
0 → 8, 2
1 → 19, 6

Post order

-1 → 9
0 → 2, 8
1 → 19, 6




HM

1 → 6, 19
0 → 8, 2
2 → 10, 4
-1 → 9, 12

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

(4,2)

```
for (i = min; i <= max; i++) {
    hm.get(i);
}
```

```
class Pair {
    Node first;
    int second;
    class Pair(x, y) {
        first = x;
        second = y
    }
}
```

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

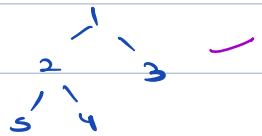
Break - 10:17 - 10:30 pm

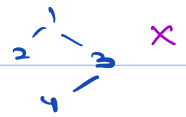
for top view,
hm.get(i).get(0)

# Types of B.T.

① Proper B.T.

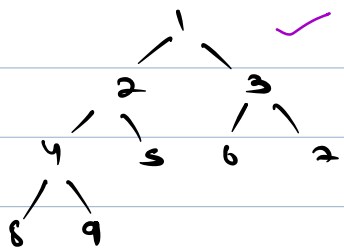
↳ ∀ nodes 0 or 2 children.

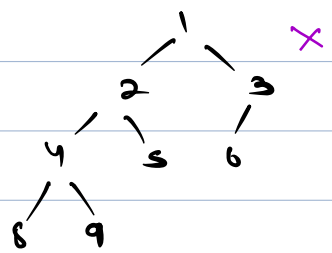


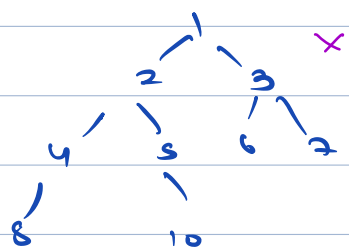


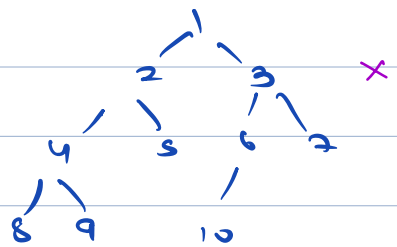
② Complete B.T.

All level must be completely filled, except may be last level but that should also be filled from L-R.

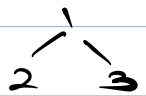


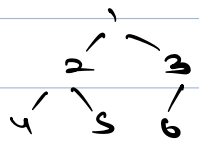






③ Perfect B.T. → All levels are completely filled.



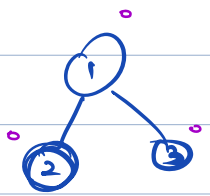



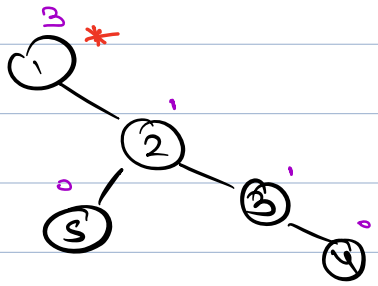
Ques **Balanced Binary Tree**

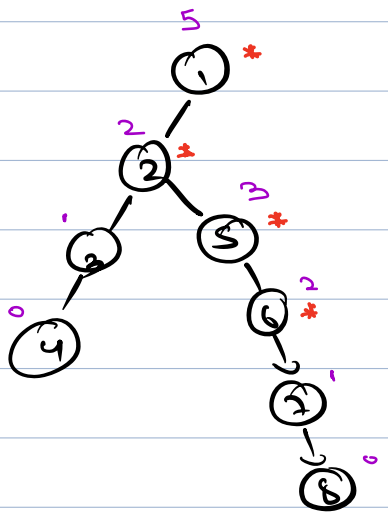
↳ A tree in which N nodes

$|LSTh - RSTh| <= 1$

**Height** in terms of nodes.

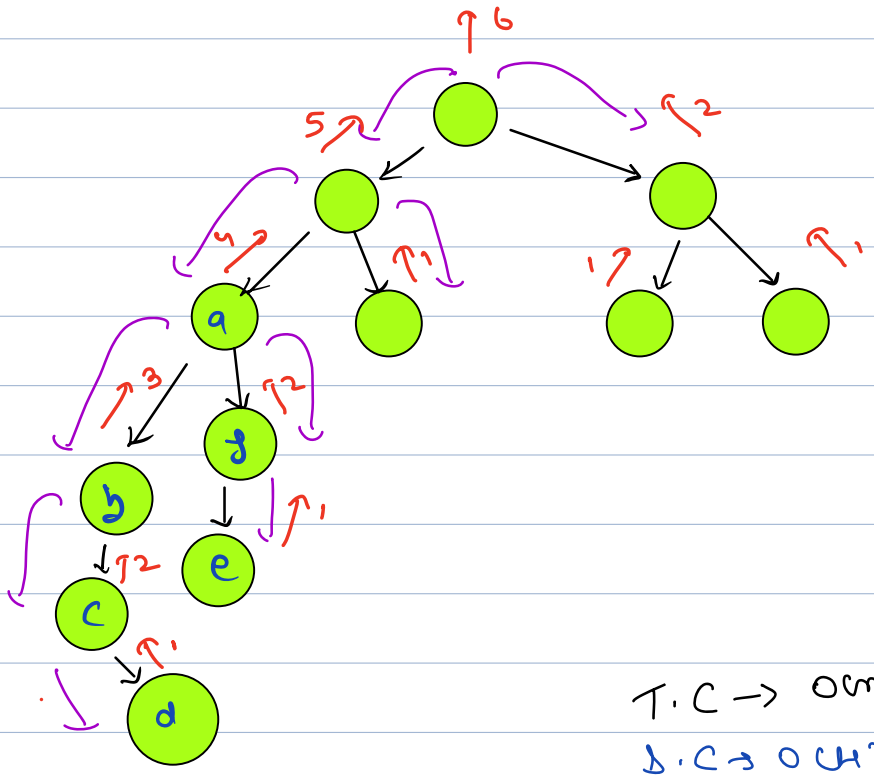



$h = \max(lh, rh) + 1$

Ques check if a tree is height balanced.



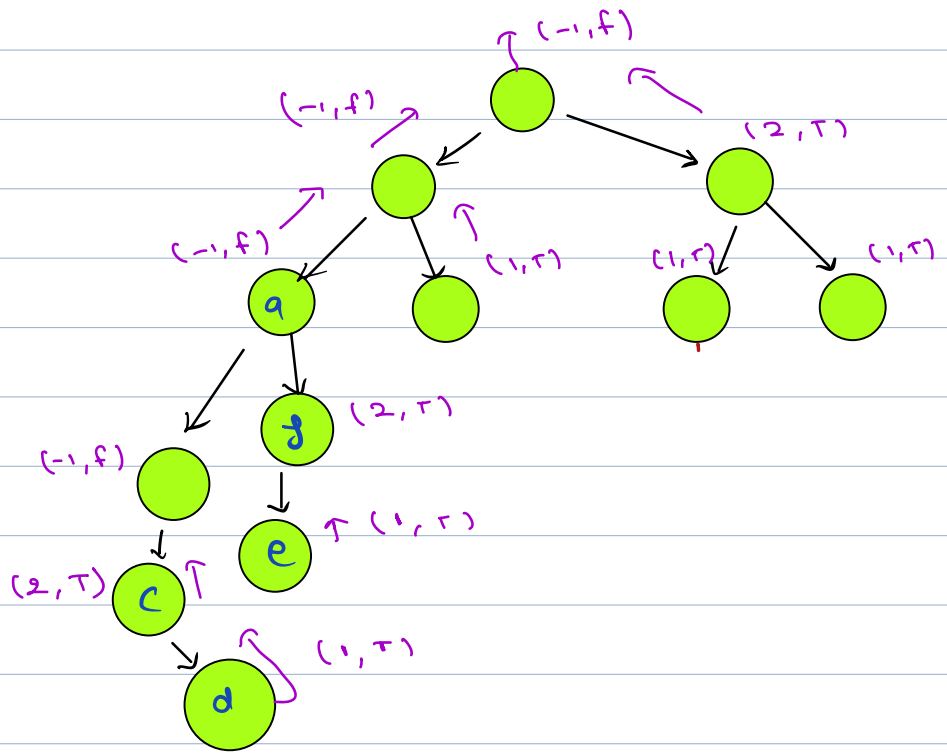

T.C → O(n)

S.C → O(H)

```
boolean isb;

boolean isBalanced(Node root) {
    isb = True;
    height(root);
    return isb;
}

int height(Node root) {
    if(root == null) {return 0}
    int lh = height(root.left);
    int rh = height(root.right);
    if(Abs(lh-rh) > 1) { isb = false }
    return Max(lh, rh) + 1
}
```




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

```
bool isBalanced ( Node root) {
    return helper(root).isBalanced;
}
```

T.C → O(n)
S.C → O(H)

```
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)
    }
}
```