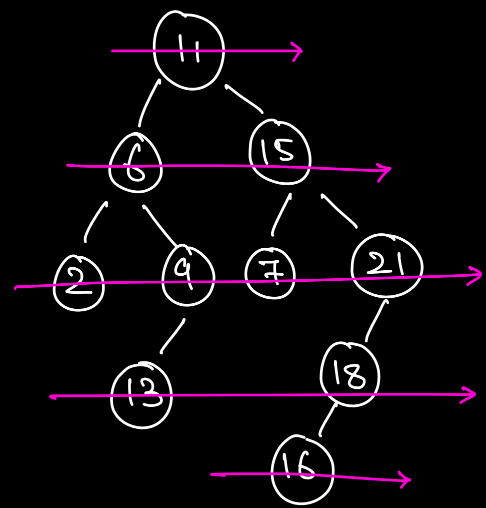
* Different type of Tree traversals & Views :

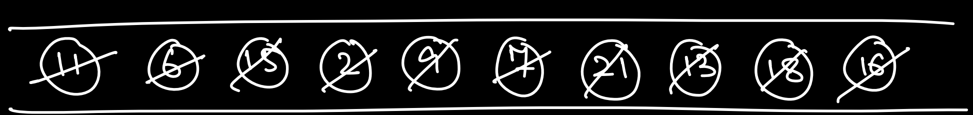
Q.1 Given a Binary Tree, Print the level order traversal.

11, 6, 15, 2, 9, 7, 21, 13, 18, 16



1

Queue : Queue <Node> q;

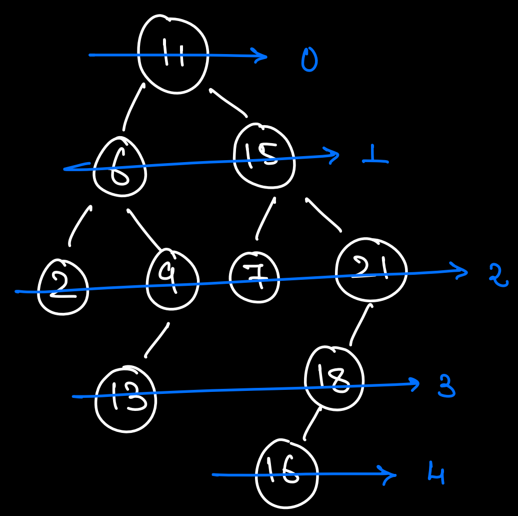


11, 6, 15, 2, 9, 7, 21, 13, 18, 16

② [[11],
[6, 15],
[2, 9, 7, 21]
[13, 18]
[16]]

List<List<int>> ans;

* Give the output in the form of list of list of int.
* All the nodes of a level will be in separate list.



① Along with the nodes we can maintain the level of the Node in the Queue.

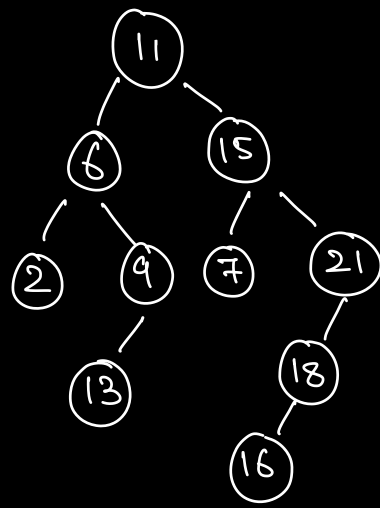
~~<11, 0>~~ ~~<6, 1>~~ ~~<15, 1>~~ ~~<2, 2>~~ ~~<9, 2>~~ <7, 2> <21, 2> <13, 3> ...

Queue<pair<Node, int>>

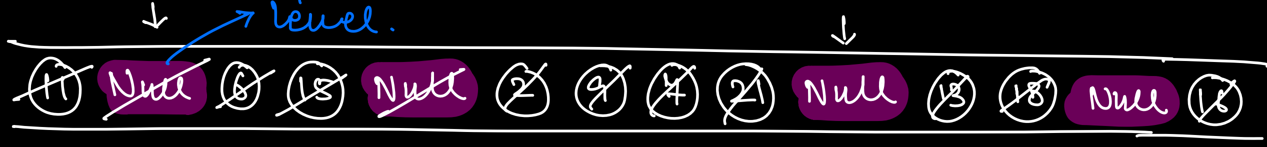
[11]
[6, 15]
[2, 9,

(1) Queue<Node> q;

* Add a marker/delimeter Node after every level.



represents a New level.

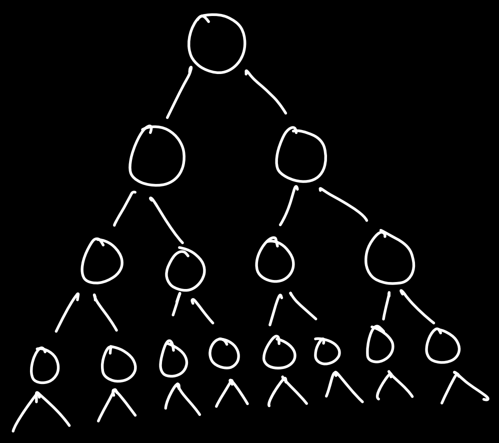


[[11]
[6, 15]
[2, 9, 7, 21]
[13, 18]
[16]]

TC: O(N)

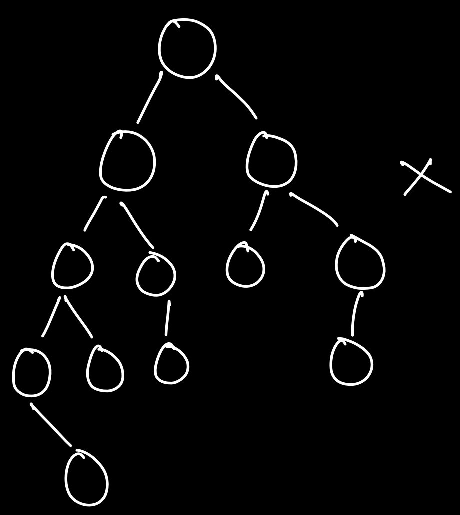
SC: Queue size

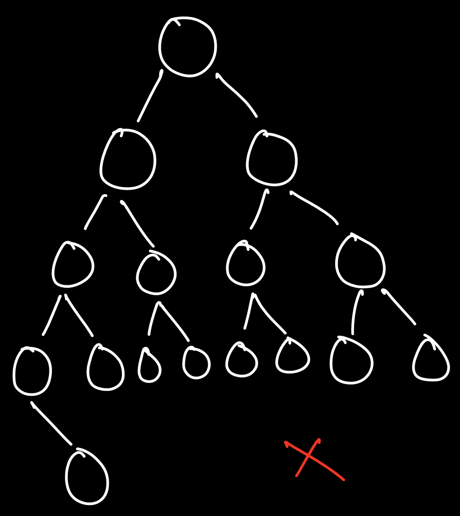
Max. no. of nodes in the Queue.

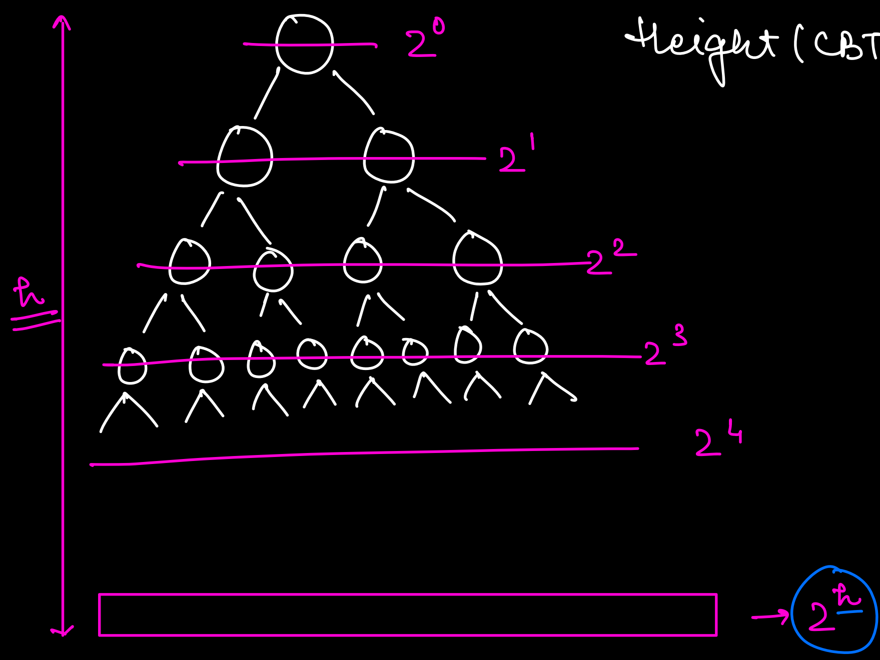


* Complete Binary Tree (CBT)

A B.T where all the levels are completely filled
↳ except possibly the last level.
→ Nodes in the last level are left aligned.







Height (CBT) $\simeq \log N$

$\Rightarrow O(\log N)$

$$\# \text{ of nodes} = 2^0 + 2^1 + 2^2 + 2^3 + \cdots 2^h$$

(Geometric Progression)

$$\text{Sum} = \frac{a(1-r^n)}{(1-r)}$$

$a = 2^0 = 1$

$r = 2$

$\#$ of terms $= h+1$

$$\# \text{ of nodes} = \frac{2^0(1-2^{h+1})}{1-2} = 2^{h+1} - 1$$

$$\# \text{ of nodes} = 2^{h+1} - 1 = N$$

$$2^{h+1} = N+1$$

$$\log_2(2^{h+1}) = \log_2(N+1)$$

$$h+1 = \log(N+1)$$

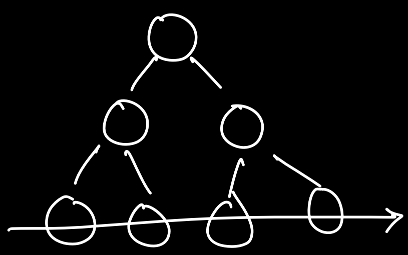
$$\boxed{h = \log(N+1) - 1} \approx \log N$$

$$\#\text{ of nodes in last level} = 2^{(\log(N+1)-1)}$$

$$\boxed{a^{\log_a x} = x}$$

$$= \frac{2^{\log_2(N+1)}}{2}$$

$$= \boxed{\frac{N+1}{2}}$$



$N = 7$

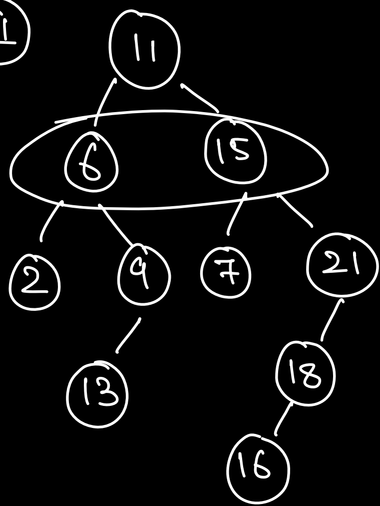
$$\frac{7+1}{2} = \frac{8}{2} = 4$$

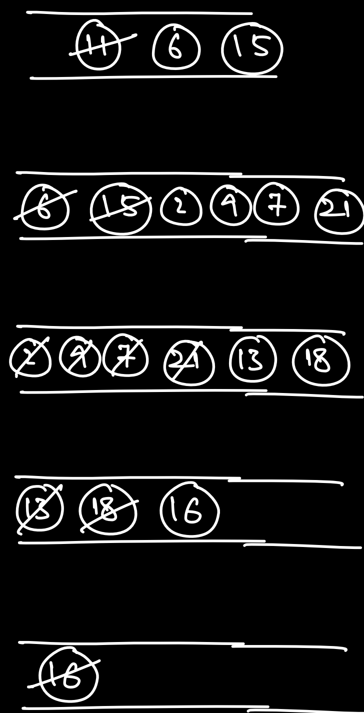
SC: Max size of the Queue

$$\Rightarrow \frac{(N+1)}{2}$$

$$\Rightarrow \boxed{O(N)}$$

III





# of dequeue operations.	Size of Queue
1	1
2	2
4	4
2	2
1	

No. of dequeue operations at each level ≡ Size of the Queue.

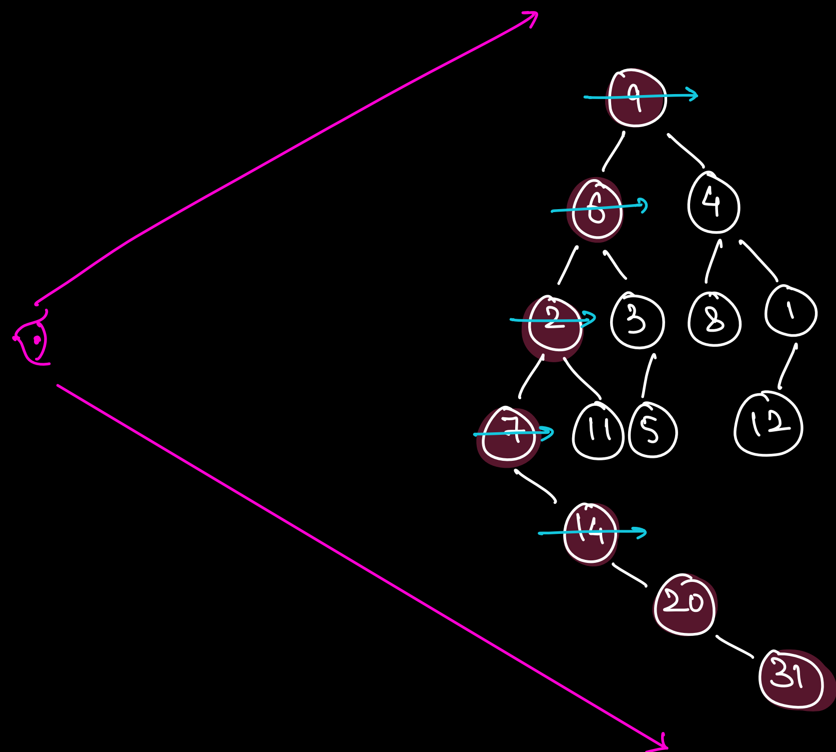
```
* list<list<int>> levelOrder(root) {
        if (root == NULL)
            return Empty list;
        Queue<TreeNode> q;
        list<list<int>> ans;
        q.enqueue(root);
        while (q.size()>0) {
            list<int> level;
            Size = Q.size();
            for(i=0; i<size; i++){
                TreeNode temp = q.front()
                q.dequeue();
                level.add(temp.data);
                if (!temp.left)
                    q.enqueue(temp.left)
                if (!temp.right)
                    q.enqueue(temp.right);
            }
            ans.add(level);
        }
        return ans;
}
```

TC: O(N)

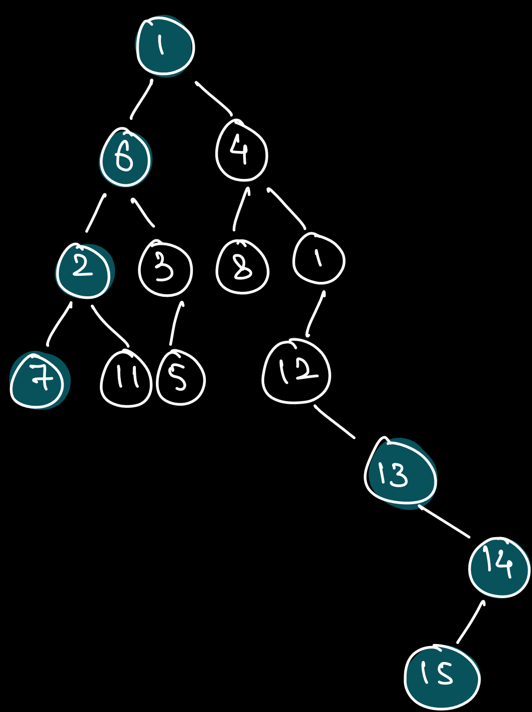
SC: O(N)

Q-2 Given a B·T, print its left View.

Amazon
MS
Adobe.

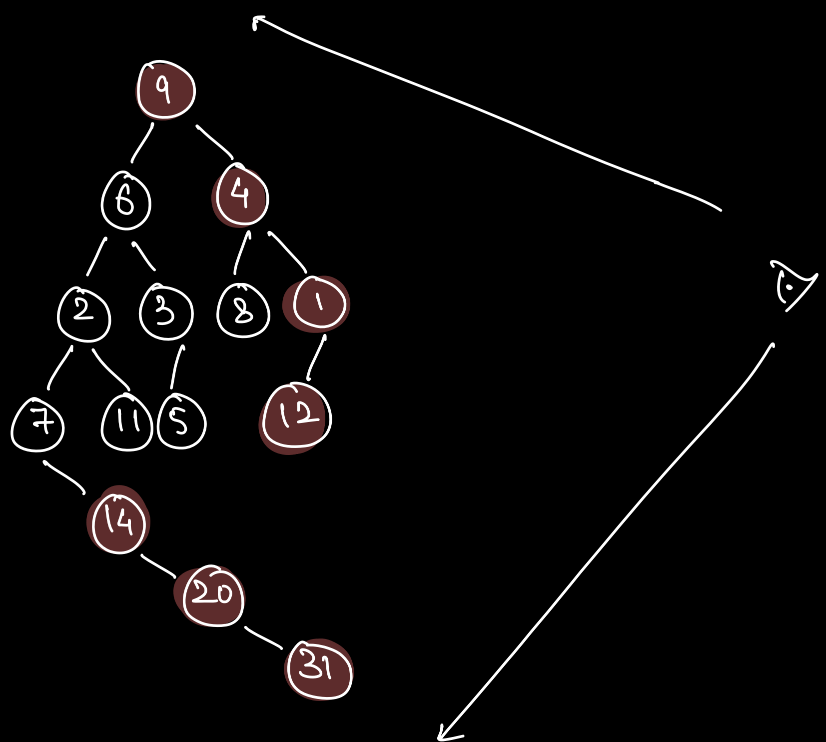


Left View : first Node of every level.



Q.2 Given a B.T, print its right View.

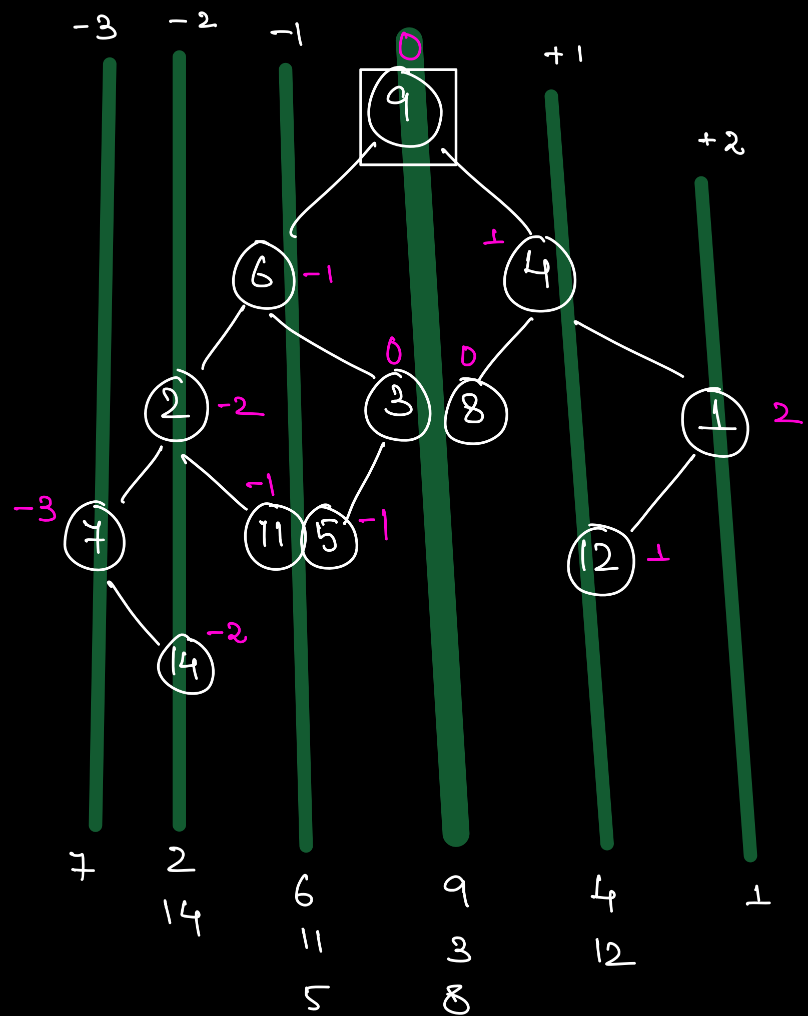
Amazon
MS
Adobe.



Right View : last node at each level.

Q. Vertical Order Traversal

[[7],
[2,14],
[6, 11, 5],
[9, 3, 8],
[4, 12],
[1]]



-3 → 7
-2 → 2, 14
-1 → 6, 11, 5
0 → 9, 3, 8
⋮

→ <k, v>

→ HashMap<int, list<int>>

HashMap< int, List< TreeNode/Int >>

distance (d) from the root.

List of nodes which are (d) distance apart from root.

→ Map : Global

```
preOrder(root, dist) {
    if (root == NULL)
        return;
    if (!map.contains(dist)) {
        map.insert(dist, new ArrayList<int>());
    }
    map.get(dist).add(root.data);
    preOrder(root.left, dist-1);
    preOrder(root.right, dist+1);
}
```

HashMap

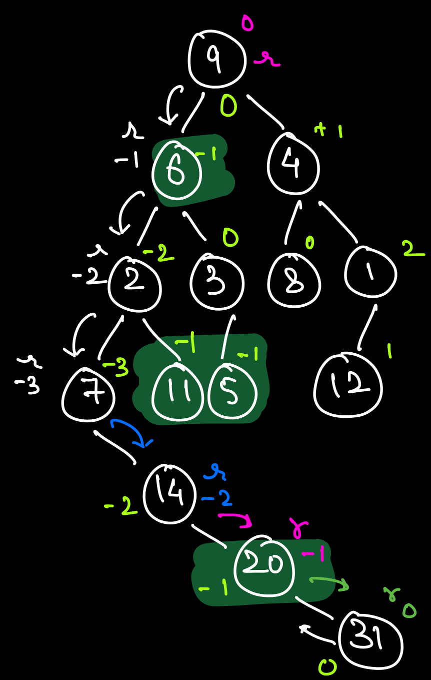
dist : List of int

0 : [9, 31

-1 : [6, 20,

-2 : [2, 14,

-3 : [7,



* PreOrder
* Inorder
* Postorder

} Dfs.

⇒ Queue : Level Order Traversal ✓

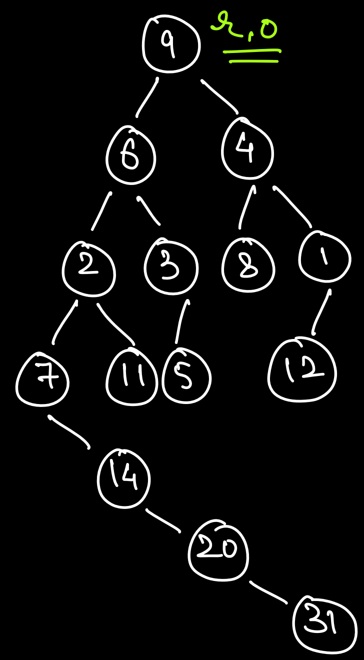
Queue < pair < TreeNode , int > >

⇑
dist

Class TreeInfo {
    TreeNode node;
    int dist;
}

Queue < TreeInfo >

Queue < TreeInfo > q;



~~{9, 0}~~ ~~{6, -1}~~ ~~{4, 1}~~ ~~{2, -2}~~ ~~{3, 0}~~ ~~{8, 0}~~ ~~{1, 2}~~

~~{7, -3}~~ ~~{11, -1}~~ ~~{5, -1}~~ {12, 1} {14, -2}

HashMap

dist : List of int

0 : [9, 3, 8,

-1 : [6, 11, 5,

1 : [4,

-2 : [2,

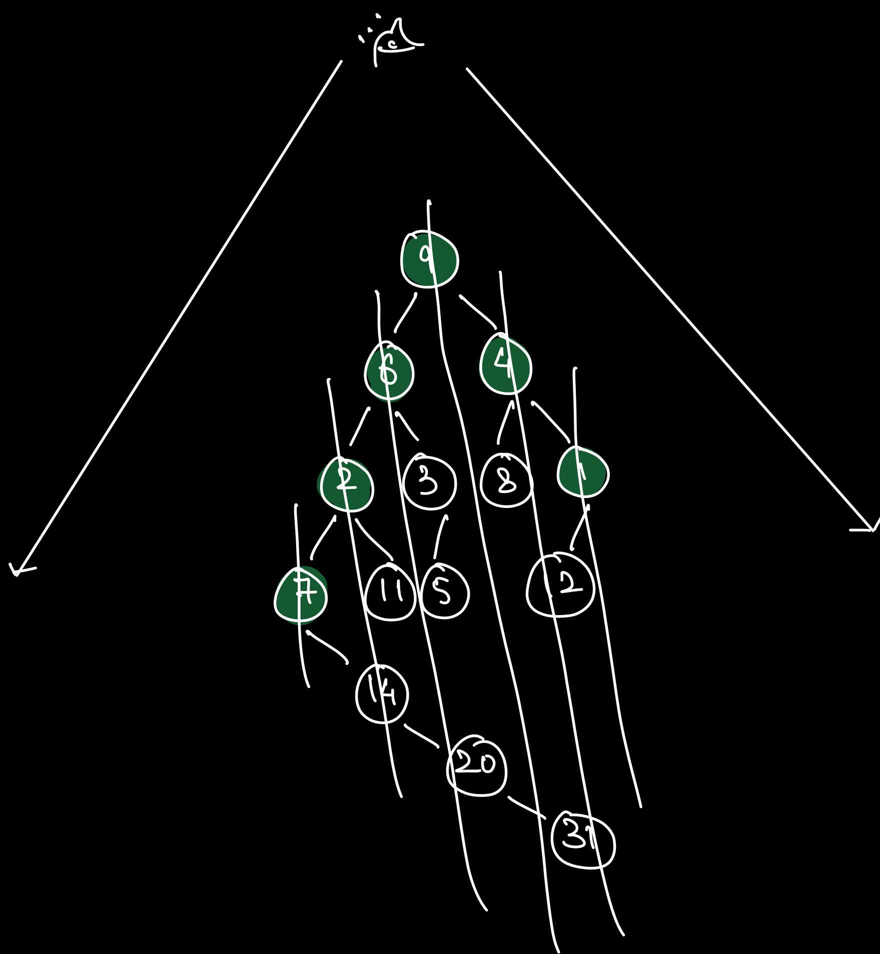
2 : [1,

-3 : [7,

min_dist → -3

max_dist → +3

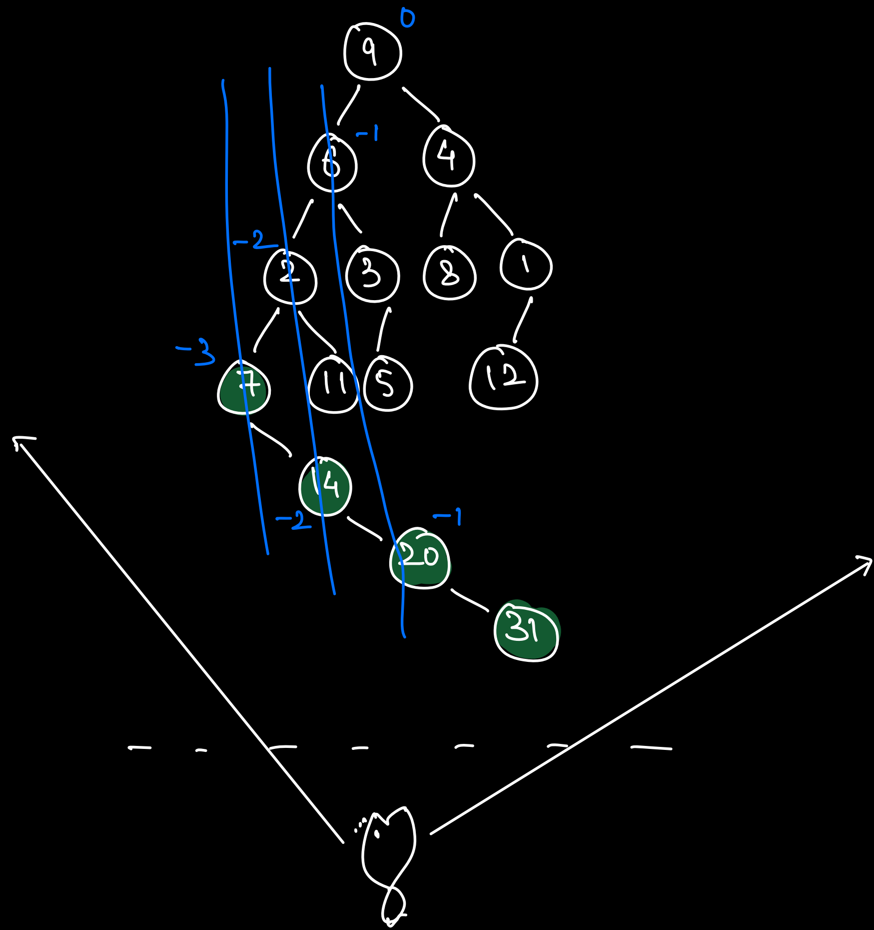
Q. Top View of Binary Tree.



⇒ First node in the list against every key (dist) in the Map.

Instead of list create single entry key->value pair in above code

Q. Bottom View of Binary Tree.



Bottom View : last node in the list against every key in the Map.

---- * ----