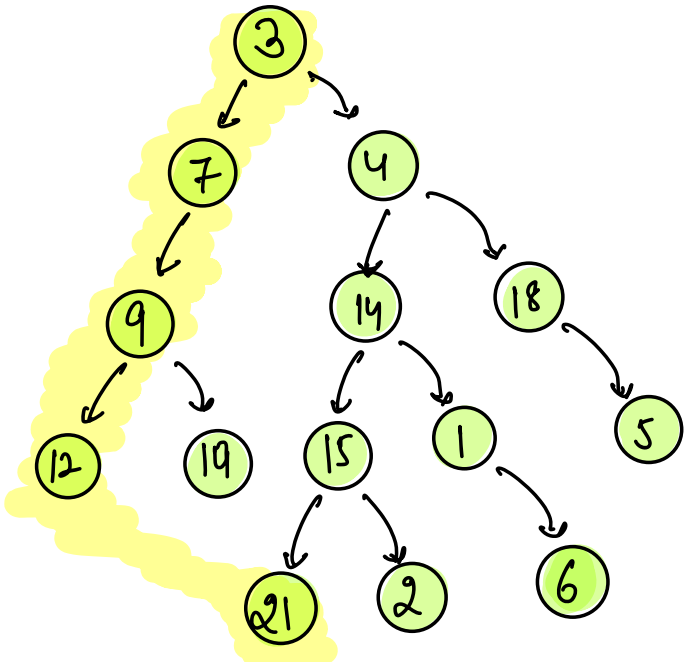
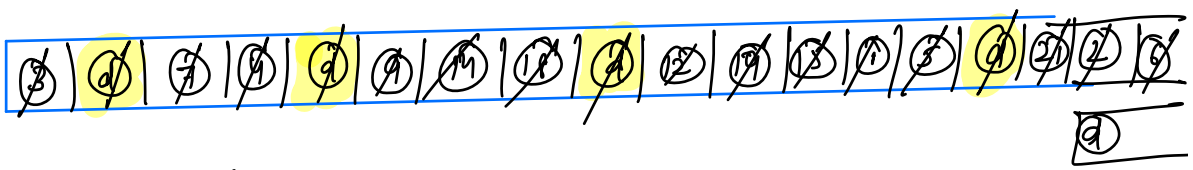
Left View.



Idea:- Print the first node of each & levery level.



o/p- [3, 7, 9, 12, 21]

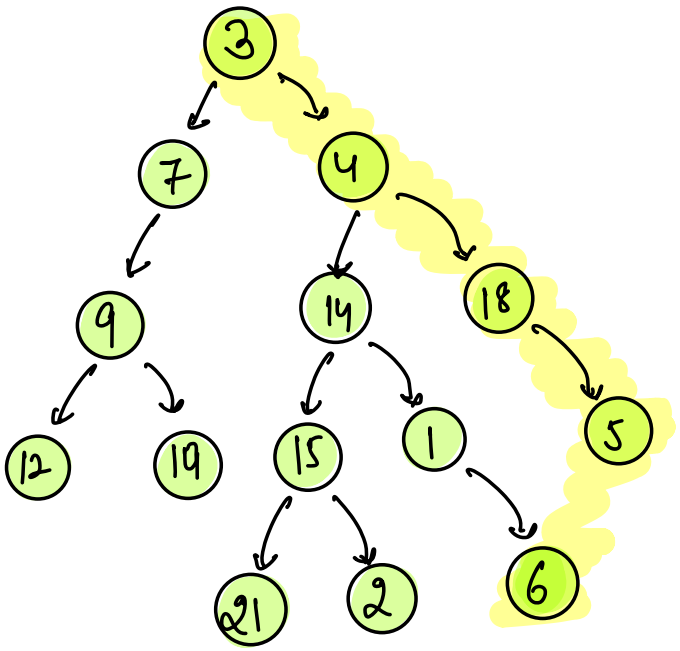
print 1st node after every dummy node in level order traversal linewise.

pseudo-code.

```
Node dummy = new Node(-1);
Queue<Node> q = new ArrayDeque<>();
q.enqueue(root); q.enqueue(dummy);
print(root.data);
while(q.size() > 1){
        Node x = q.dequeue();
        if(x == dummy){
                print(q.front().data);
                q.enqueue(dummy);
        }else{
                if(x.left != NULL){ q.enqueue(x.left) }
                if(x.right != NULL){ q.enqueue(x.right) }
        }
}
```

[T.C → O(N)
S.C → O(N)]

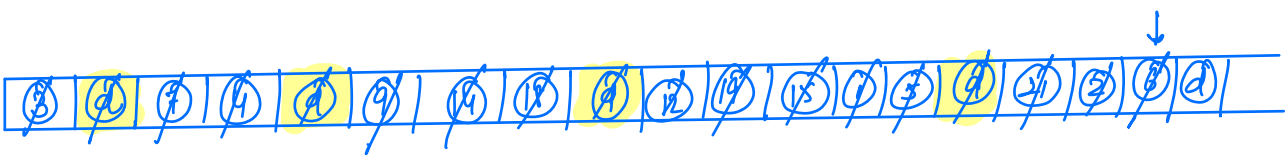
Right View



idea → print the last element of each level.

↓

element present just before dummy node.



Node prev → NULL.

o/p → [3, 4, 18, 5, 6]

#pseudo-code - 1.

```
Node dummy = new Node(-1);
Queue<Node> q = new ArrayDeque<>();
q.enqueue(root); q.enqueue(dummy);
Node prev = NULL;
while (q.size() > 1) {
    Node x = q.dequeue();
    if (x == dummy) {
        print(prev.data);
        q.enqueue(dummy);
    } else {
        if (x.left != NULL) { q.enqueue(x.left) }
        if (x.right != NULL) { q.enqueue(x.right) }
    }
    prev = x;   // update prev Node just before next iteration.
}
print(prev.data); // ?
```

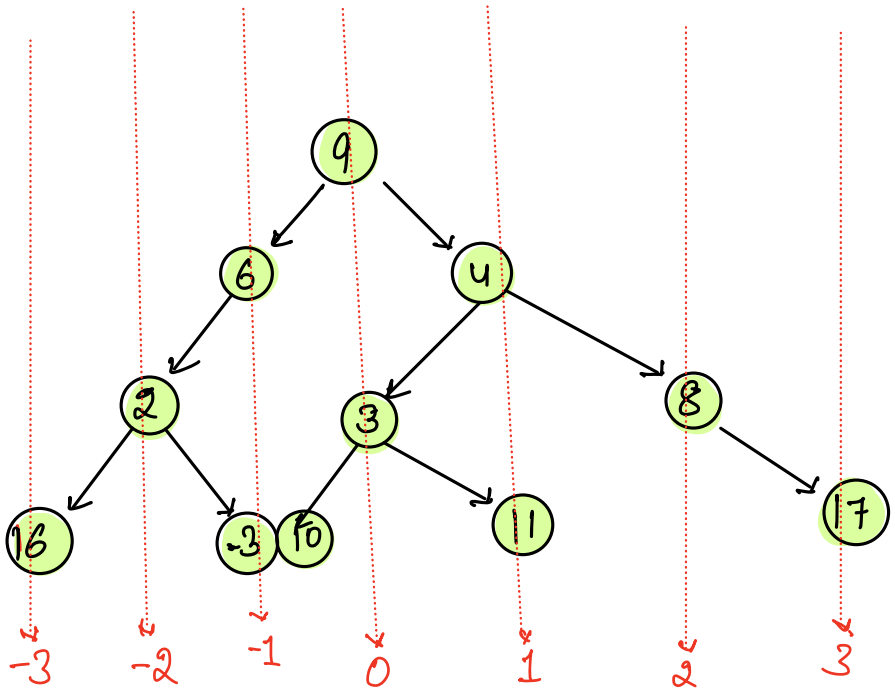
[T.C → O(N)
S.C → O(N)]

pseudo code 2:

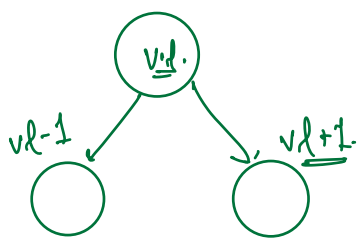
```
Node dummy = new Node(-1);
Queue<Node> q = new ArrayDeque<>();
q.enqueue(root); q.enqueue(dummy);
print(root.data);
while (q.size() > 1) {
        Node x = q.dequeue();
        if (x == dummy) {
            print(q.front().data);
            q.enqueue(dummy);
        } else {
            if (x.right != NULL) { q.enqueue(x.right) }
            if (x.left != NULL) { q.enqueue(x.left) }
        }
}
```

[T.C → O(N)
S.C → O(N)]

Vertical Order Traversal (top to bottom)



o/p → [16 / 2 / 6 -3 10 / 9 3 / 4 11 / 8 / 17]

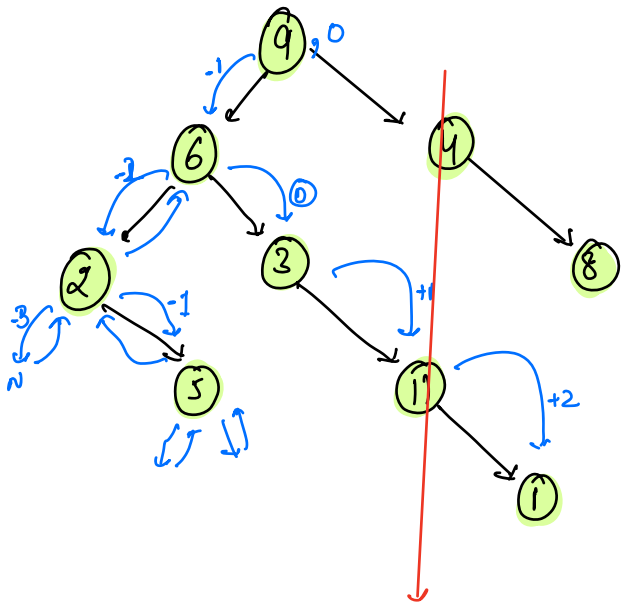


Hashmap

{
0 → 9, 3
-1 → 6, -3, 10
-3 → 16
-2 → 2
3 → 17
2 → 8
1 → 4, 11
}

→ pre-order

→ level-order

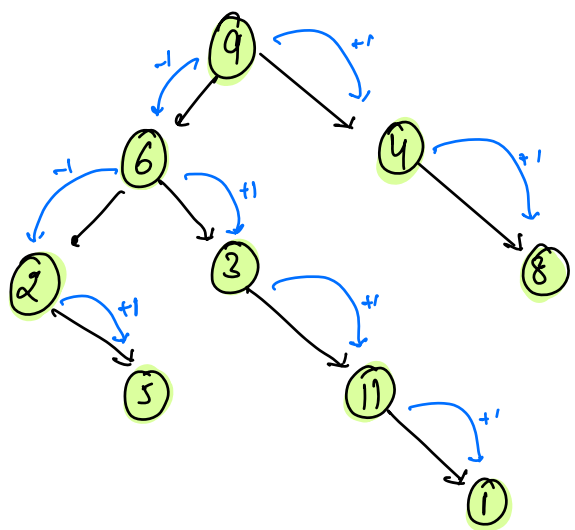


H.m.

0	9, 3
-1	6, 5
-2	2
+1	11
+2	1

int → list<Node>

pre-order can't help you with top-bottom.



int	list<Node>
0	9, 3
-1	6, 5
1	4, 11
-2	2
2	8, 1

9,0	6,-1	4,1	2,-2	3,0	8,2	5,-1	11,1	1,2

minVl = -2, maxVl = +2.

```
for( i = minVl ; i <= maxVl; i++){
    get the corresponding data
        of i from HM
}
```

pseudo-code -

Pair → (Node, level)

```
HashMap < integer , List<Node>> hm;
Queue < Pair > q ;
q.enqueue ( {root, 0} );
while( q.size() > 0){
        Node rp = q.dequeue();
        if ( hm.containsKey(rp.level)){
                hm[rp.level].add(rp.node);
        }else{
                List<Node> l ;
                l.add(rp.node);
                hm.insert ( rp.level , l);
        }                 key        value (list<Node>)
        // insert left child {rp.node.left, rp.level -1}
        // insert right child {rp.node.right, rp.level +1}
        minvl = min( minvl , rp.level)
        maxvl = max( maxvl , rp.level)
}
for( i = minvl ; i <= maxvl ; i++){
        // print the corresponding list.
}
```

T.C → O(N)
S.C → O(N)

Top view → first node of each vertical level.

Bottom view → last node of each vertical level.

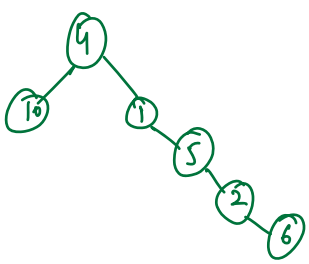
[#todo]

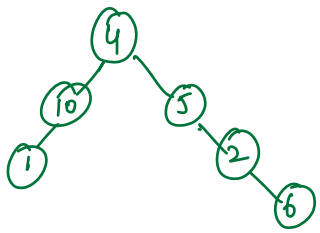
Construct Binary Tree.

① pre-Order → Node left right.

4 10 1 5 2 6

root





② post-Order left right Node.

4 10 1 5 2 6.

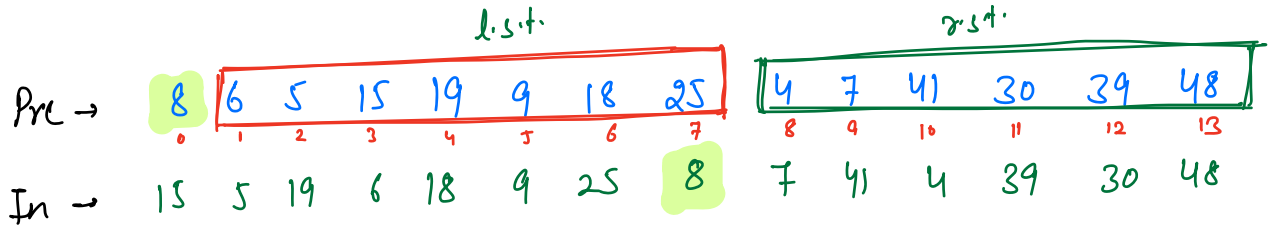
root.

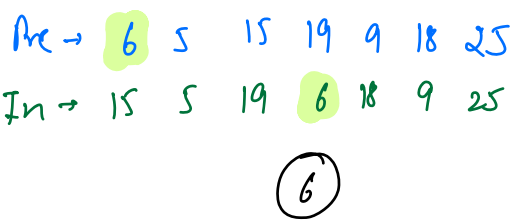
③ In-order left. Node. right.

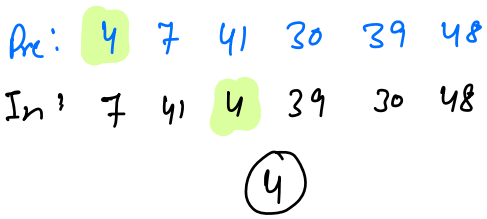
4 10 1 5 2 6.

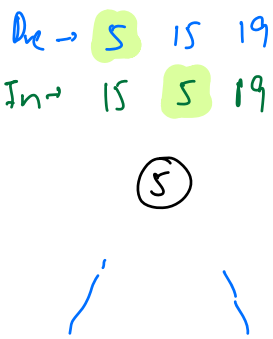
[pre + In ✓
post + In ✓]

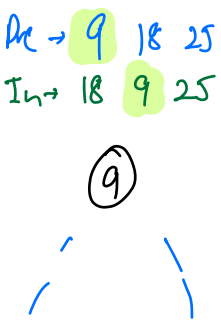
- N. distinct elements.

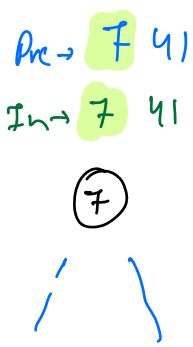


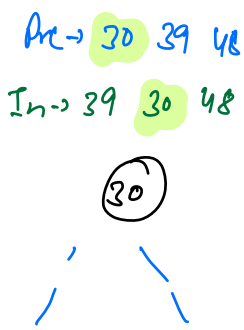


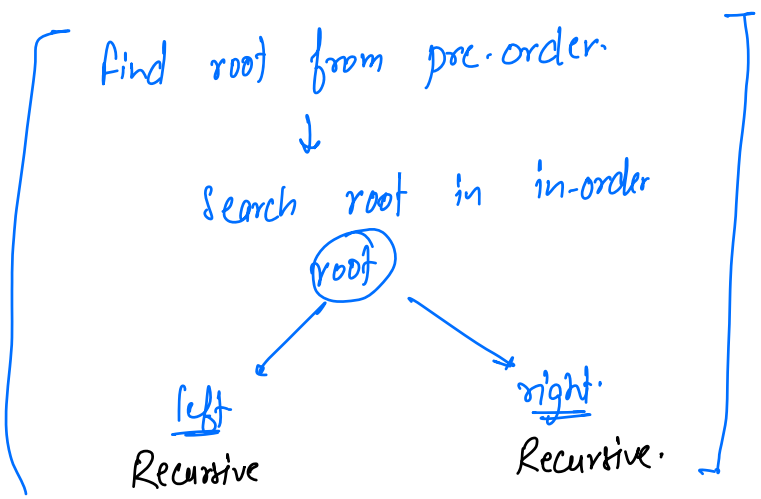








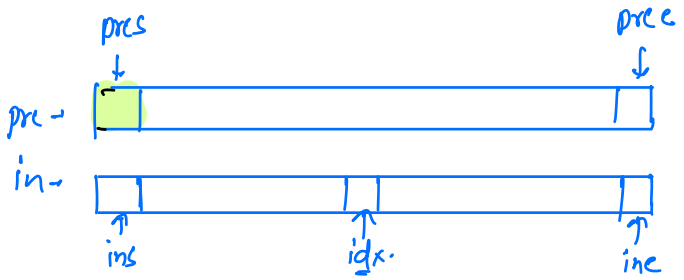




Assumption: Construct the tree & return the root.

0 N-1 0 N-1

```
Node construct (pre[], in[], prs, pre, ins, inc) {
    if (prs > pre) { return NULL }
    int val = pre[prs];
    Node root = new Node(val);
    //search root in in-order (HashMap)
    int idx = -1;
    for( i = ins; i <= inc; i++) {
        if ( in[i] == val) {
            idx = i, break
        }
    }
    l = idx - ins;
    root.left = construct ( pre, in, prs+1, prs+l, ins, idx-1);
    root.right = construct (pre, in, prs+l+1, pre, idx+1, inc);
    return root;
}
```



no. of elements in list = [ins, idx)

= idx − ins.

T.C → {# todo}
S.C →