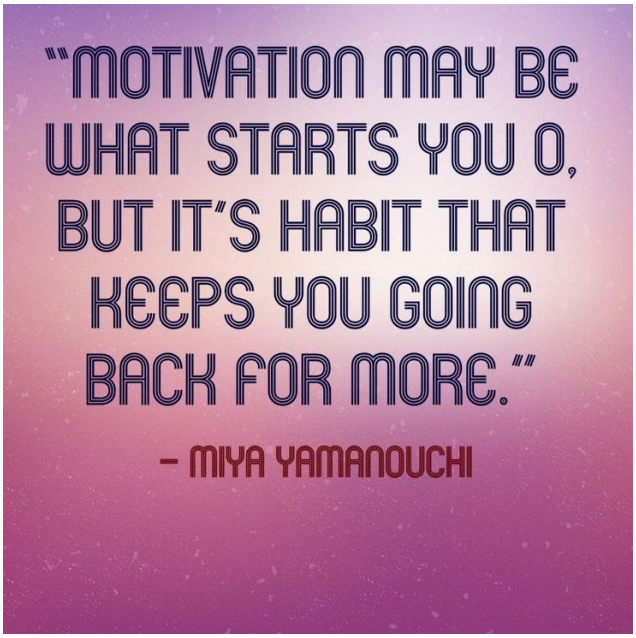
TODAY:

Trees 2

DON'T FORGET



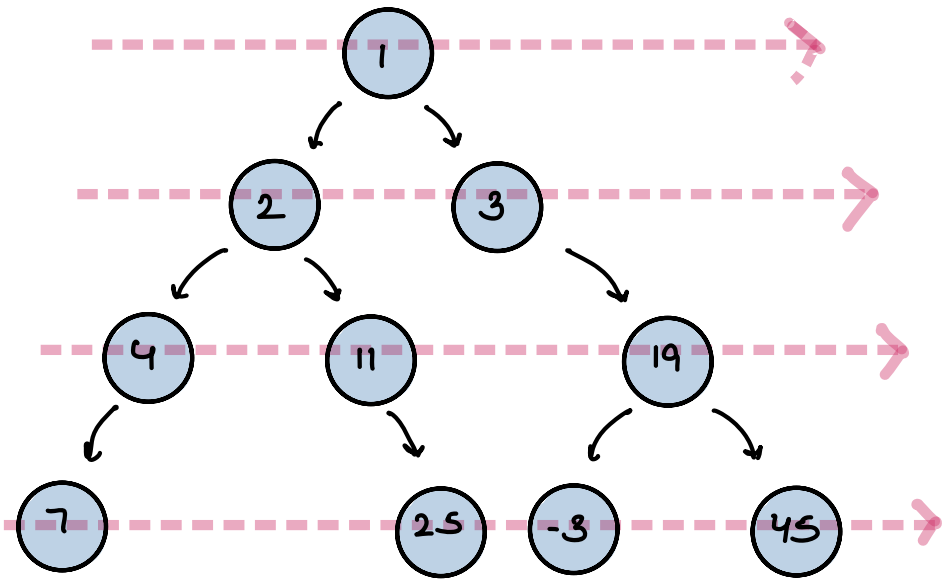
Good Evening

AGENDA:

Topics for Today

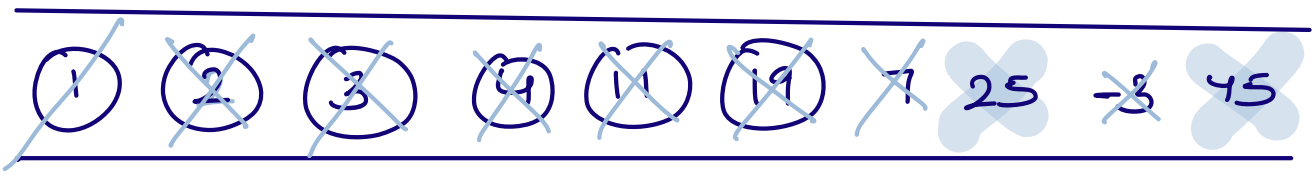
01. Level Order Traversal
02. Left view & Right view
03. Vertical level order Traversal
04. Top view & Bottom view
05. Types of B.T
06. Check height balanced

Level Order Traversal (BFS)



Ans = 1 2 3 4 11 19 7 25 -3 45

Queue →

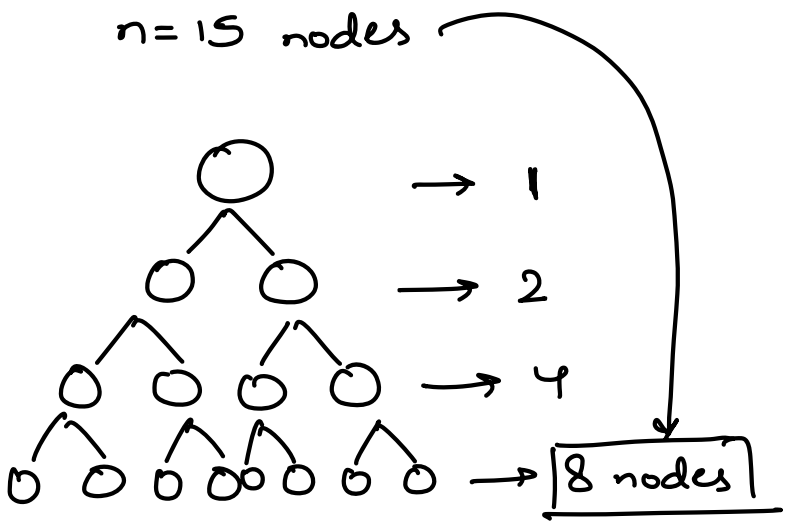


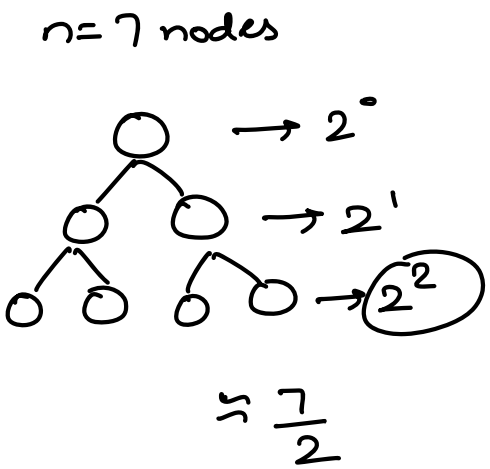
- remove
- work
- push the child

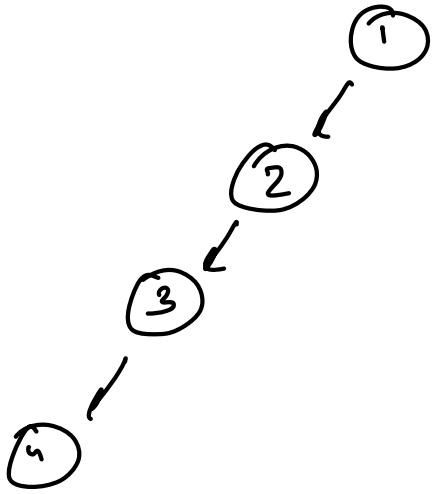
```
Queue<Node> q;

q.enqueue(root);

while(q.size()>0)
    Node rem = q.dequeue();
    print(rem.data);
    if(rem.left!=null)q.enqueue(rem.left);
    if(rem.right!=null)q.enqueue(rem.right);
```

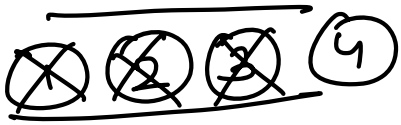
TC : O(N)

SC : O(N)





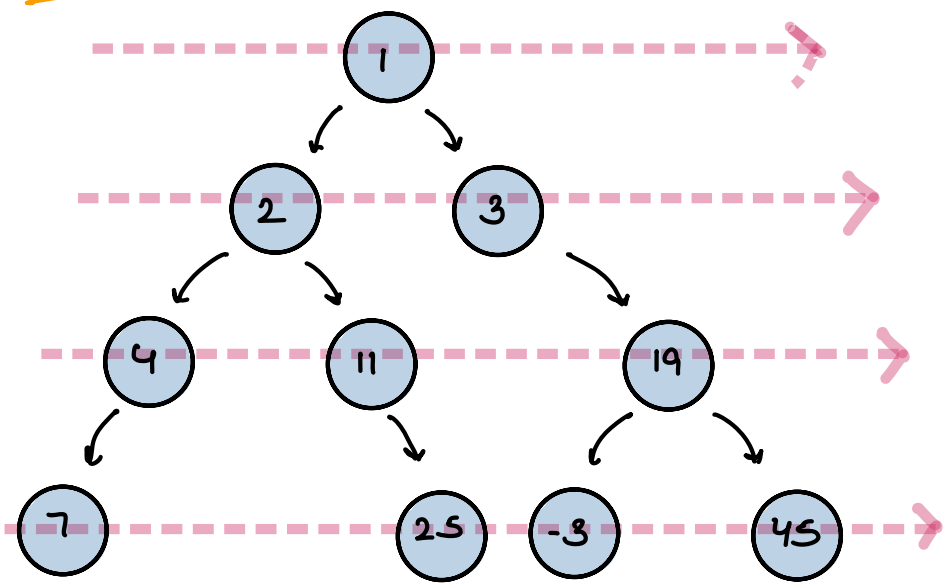




1 2 3

SC : O(1)

Q2



Output

1

2 3

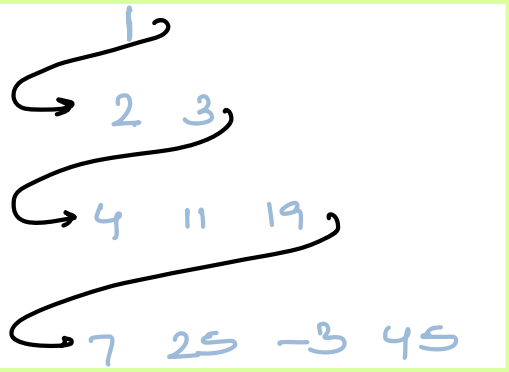
4 11 19

7 25 -3 45

counter for nodes at a particular level

Queue

Count = 1 | 2 1 0 | 3 2 1 0 | 4 3 2 1 0



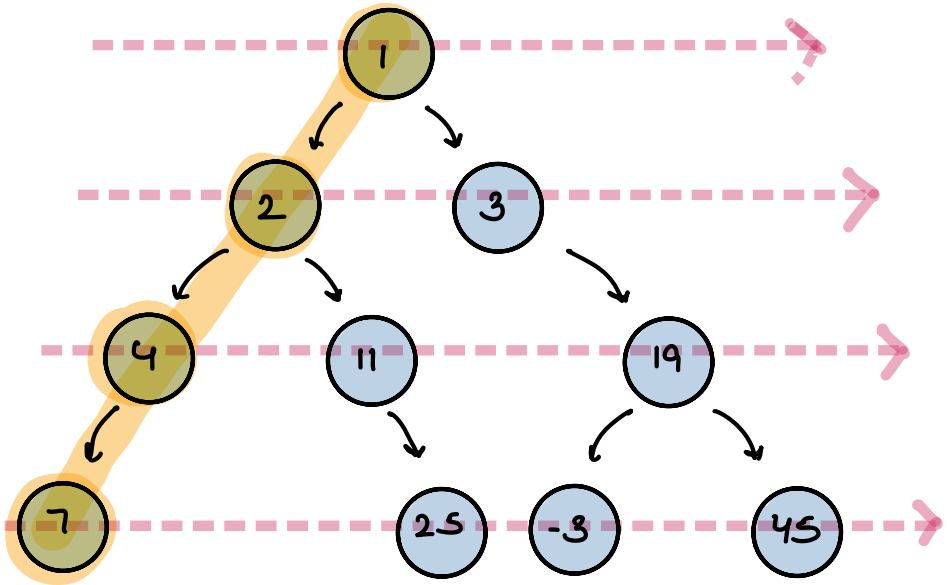
← Output

```
Queue<Node> q ;

q.enqueue(root);

while (q.size()>0)
    int sz = q.size();
    for ( i=1; i≤ sz ; i++){
        Node rem = q.dequeue();
        print(rem.data);
        if (rem.left!=null) q.enqueue(rem.left);
        if(rem.right!=null) q.enqueue(rem.right);
    }
    println();
}
```

TC: O(n)

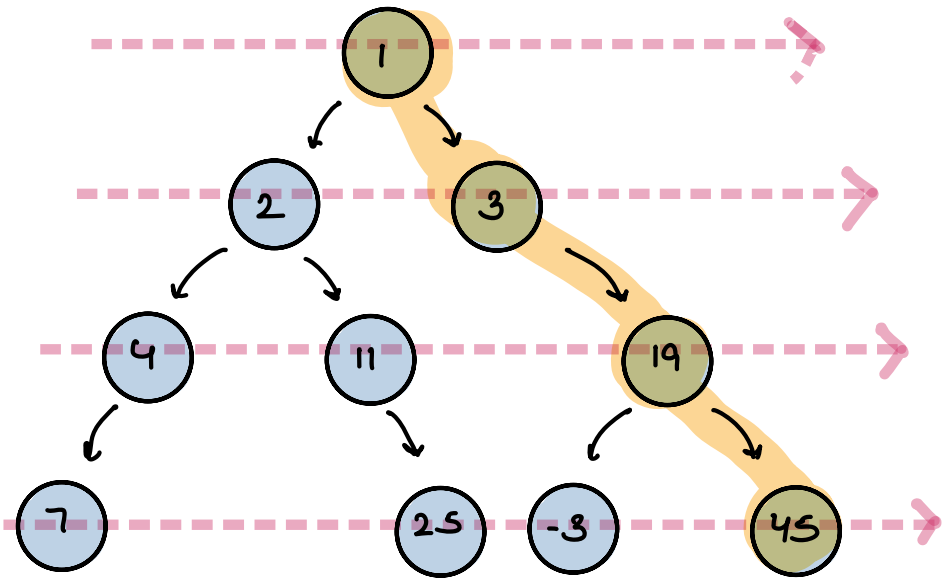
SC: O(n)

* Left view of tree



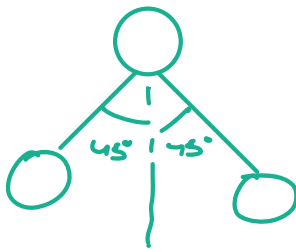

```
Queue<Node> q ;

q.enqueue(root);

while (q.size()>0)
    int sz=q.size();
    for ( i=1; i≤ sz ; i++){
        Node rem = q.dequeue();
        if (i==1)   print(rem.data);
        if (rem.left!=null) q.enqueue(rem.left);
        if(rem.right!=null) q.enqueue(rem.right);
    }
    println();
}
```

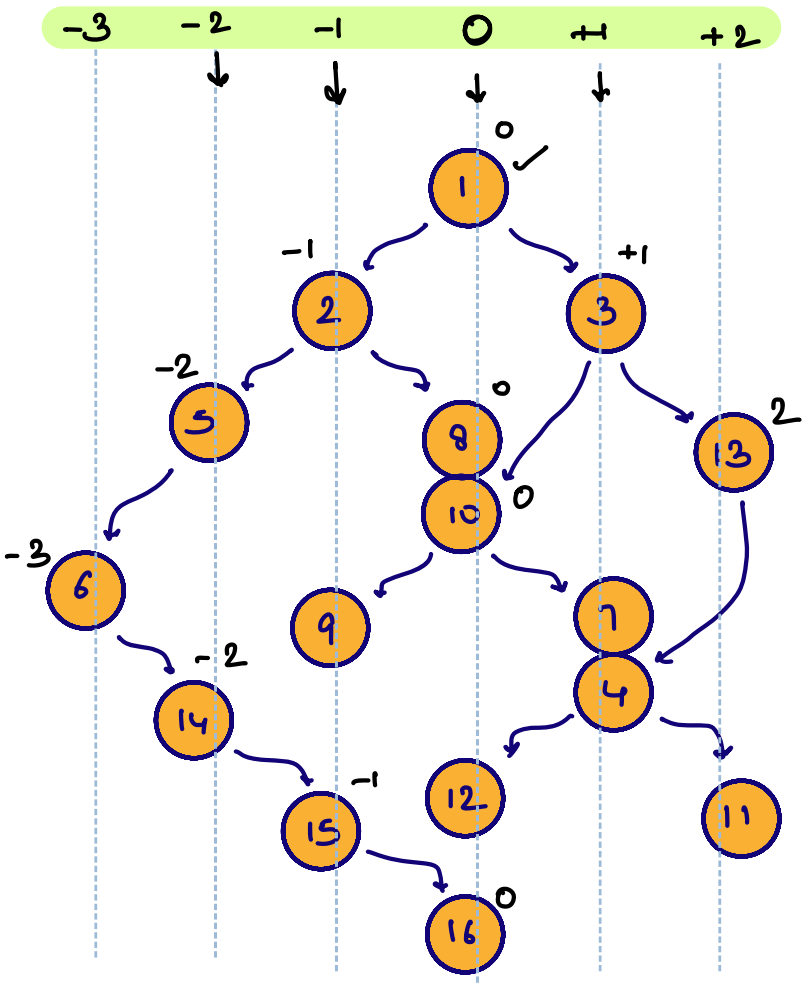
* Right view of binary tree




```
Queue<Node> q ;

q.enqueue(root);

while (q.size()>0)
    int sz=q.size();
    for ( i=1; i≤ sz ; i++){
        Node rem = q.dequeue();
        if (i==sz) print(rem.data);
        if (rem.left!=null) q.enqueue(rem.left);
        if(rem.right!=null) q.enqueue(rem.right);
    }
    println();
}
```

* Vertical Order Traversal (Microsoft)





Output

6

5 14

2 9 15

1 8 10 12 16

3 7 4

13 11

Recursion (DFS)

Iteratively (BFS)

HM

vl	Nodes
0	→ 1, 8, 10, 12, 16
-1	→ 2, 9
1	→ 3, 7, 4
-2	→ 5, 14
2	→ 13, 11
-3	→ 6

1,0 2,-1 3,1 5,-2 8,0 10,0 13,2 6,-3 9,-1

7,+1

4,1

pair = node, vl

10,0

left = 9,-1

right = 7,+1

Code

```
class Pair {
    Node node
    int vl;
}
```

3

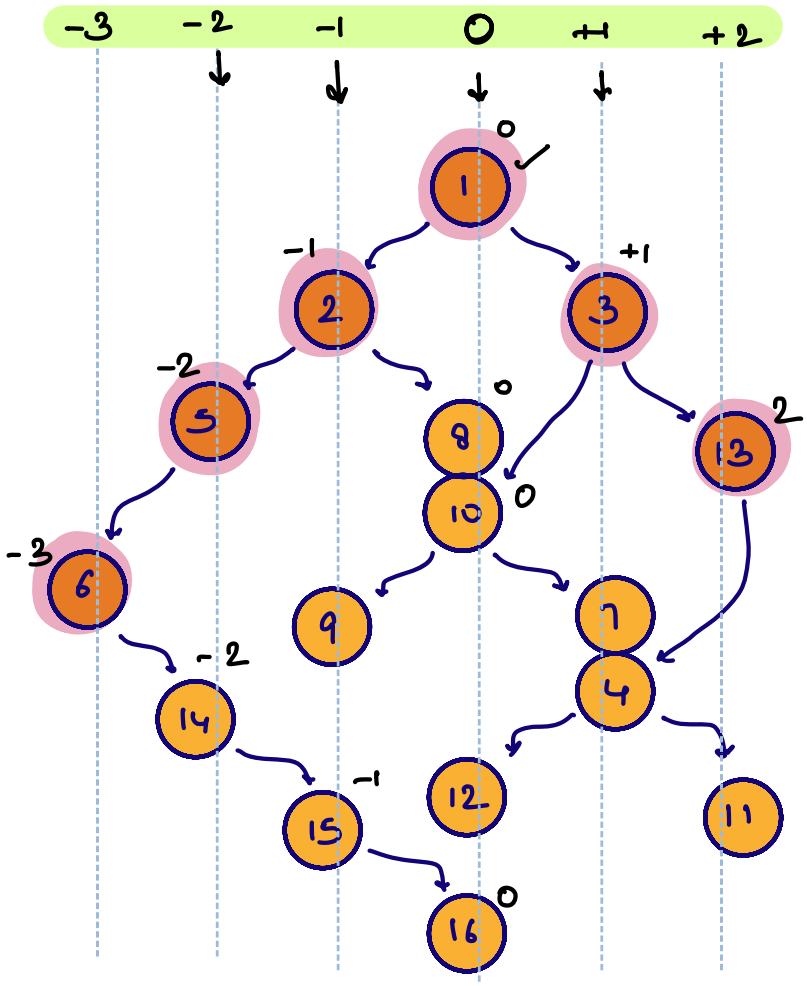
```
Pair rootpair = new pair(root,0)

Queue < pair > q ;

HashMap < int , list > map;

minvl = 0

maxvl = 0

q. enqueue (root pair):
```

TC : O(n)

SC : O(n)

```
while (q.size()>0)
{
    pair rp = q.deque();
    minvl = Min (minvl, rp.vl)
    maxvl = Max (maxvl, rp.vl);
    // Add rp.node at rp.vl ;
        hm.get(rp.vl). add( rp.node)    // Handle this carefully
    if (rp.node.left != null)
    {
        q. enque( new pair (rp.node.left, rp.vl-1);
    }
    if (rp.node.right != null)
    {
        q. enque (new pair (rp.node.right, rp.vl+1);
    }
}

for ( i= minvl ; i ≤ maxvl ; i++){
    print (map[i]);
}
```

* Top view



HM

vl	Nodes
0 →	1, 8, 10, 12, 16
-1 →	2, 9, 15
1 →	3, 7, 4
-2 →	5, 14
2 →	13, 11
-3 →	6

Top view → print first ele of every level

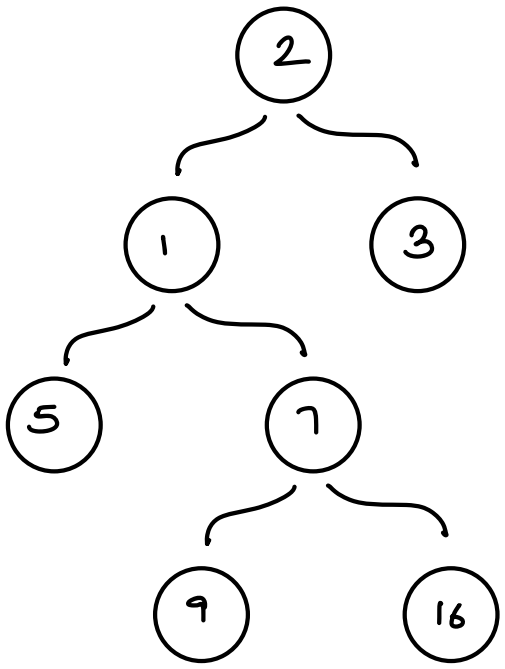
Bottom view → print last ele of every level

10:13 pm → 10:23 pm

Types of Binary Tree

01. Proper / Full Binary Tree

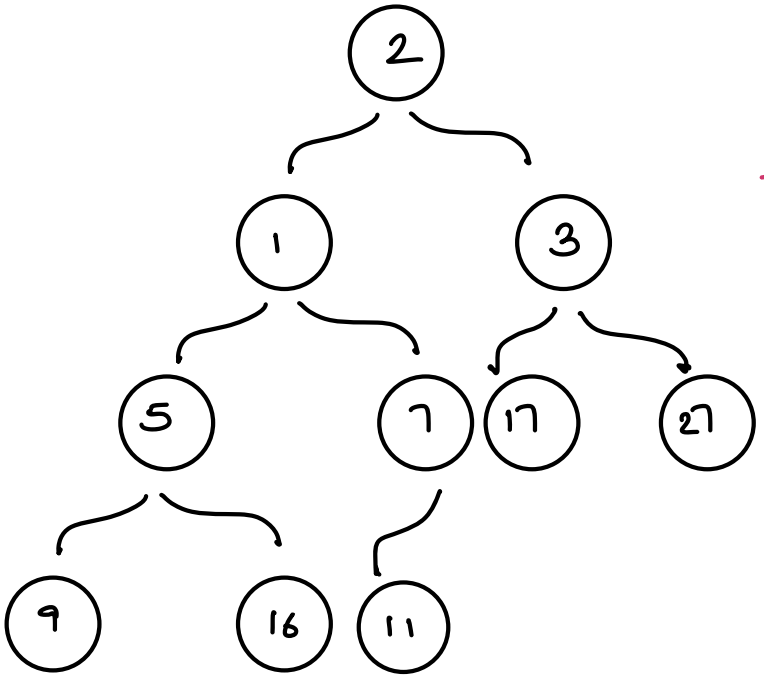
For every node, either 0 or 2 children



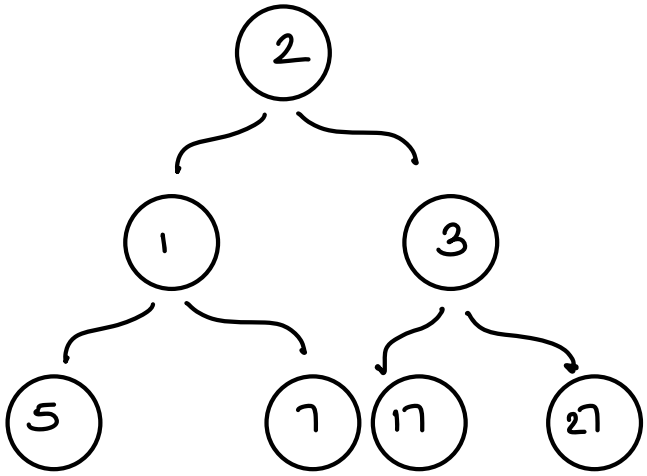
* Complete Binary Tree

→ Every level is completely filled except the last level which may or maynot be completely filled

→ In last level, nodes should be filled from LHS to RHS



* **Perfect Binary Tree** → All levels are completely filled



Are all perfect binary Tree also proper? Yes

Are all perfect BT, also complete tree? Yes

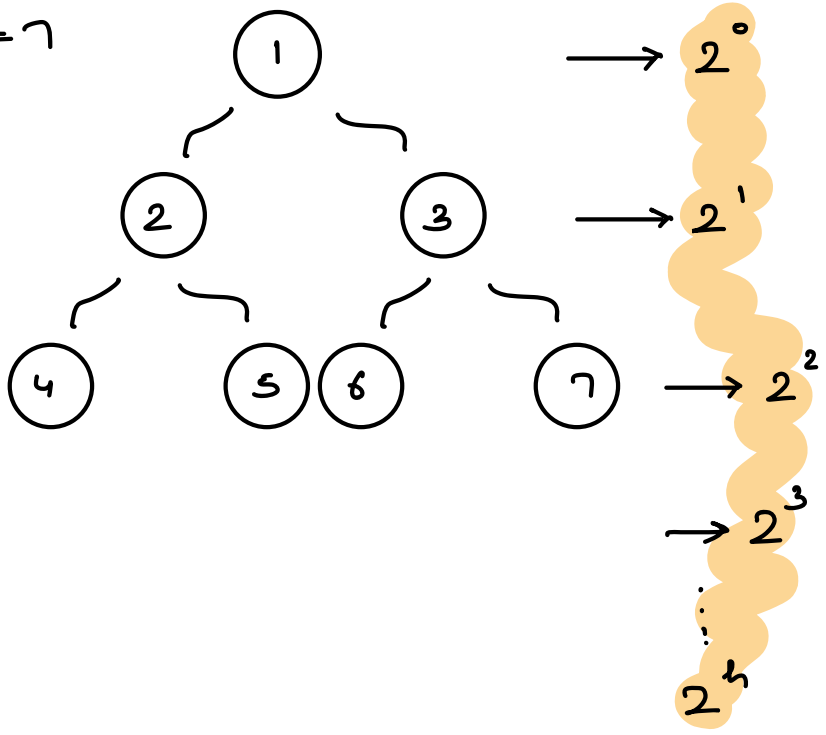
Are all complete BT, also proper tree? No

Q Given perfect BT with N nodes, find the height of tree

N = 1

h = 0

N = 7



$$\underbrace{2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^h}_{\text{no. of ele} = h+1} = N$$

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

$$\frac{+\left[2^{h+1} - 1\right]}{2 - 1} = N$$

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

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

$$h+1 = \log_2(N+1)$$

$$h = \log_2(N+1) - 1$$

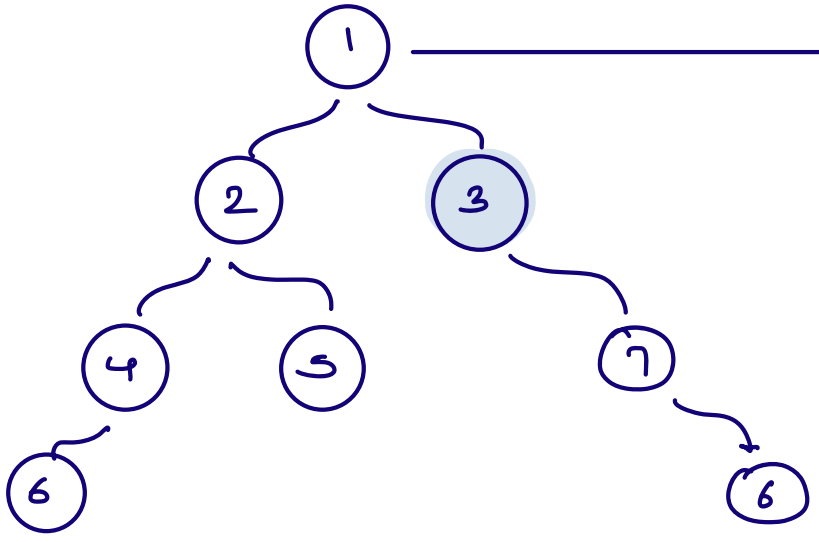
$$O(\log_2 n) \leq h \leq O(n)$$

* Balanced Binary Tree

For all nodes

$$\left| \text{height of left sub} - \text{height of righ sub} \right| \leq 1$$

Q Given a BT, check if it is balanced or not



lh & rh

* can be solved by asking recursion to return a pair

h, isbalanced

Travel & Change

```
static boolean isbalanced = true;

int height (Node root)
    if (root == null) { return -1 }
    lh = height (root.left);
    rh = height (root.right);
    if ( abs (lh - rh) > 1 ) isbalanced = false;
    return max(lh, rh) + 1;
}
```