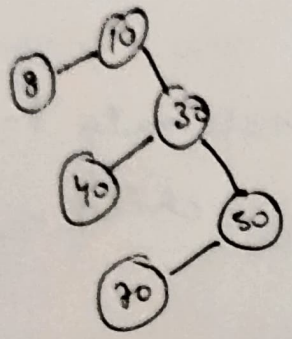
Problems:

Q. Find Height of Binary Tree.



O/P : 4

Height can be by node (or) by edges also.
↳ i.e (nodes − 1).

→ WKT, height will be the longest path in tree.
In this question we are considering nodes.

→
```
int height (Node root)
{ if (root == null)
      return 0;
  else
    { return (Math.max (height(root.left), height(root.right)) + 1);
    }
```

Complexity
↳ O(n)

O(H) → space.

Q Printing lvl order traversal.

Naive.

→ Find height of tree. → Step 1

Printing nodes at distance k from root → Step 2.

→ $O(n + Hn) \longrightarrow O(Hn)$

(height)

Eff. Using Queue DS instead of recursion. $O(n)$, $O(n)$ (or $\Theta(w)$ (width))

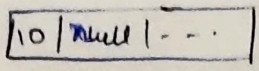
→ Java →

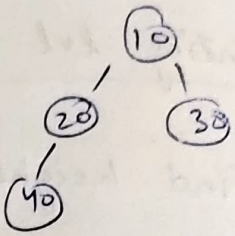
```
void printLevel (Node root)
{ if(root == null) return;
  Queue<Node> q1 = new LinkedList<Node>();
  q1.add(root);
  while(q1.isEmpty() == false)
  { Node curr = q1.poll();
    if(curr.left != null)
      { q.add(curr.left); }
    if(curr.right != null)
      { q.add(curr.right); }
  }
}
```

If we want to print line by line, we use the idea that when last elem of a lvl is poll(), the queue at that time has all numbers of next level, we put null after them.

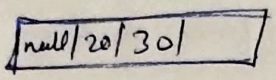
Now when this level will end, we will encounter null and repeat this process.

Eg:-

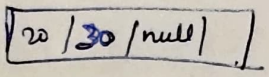




Print ten. & push next lvl.



We encounter null so we print a line & add null to know when we will print line next time.




```
if (curr == null)
{ System.out.println(null);
  q.add(null);
} continue;
```

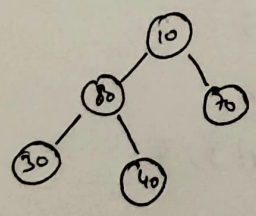
Use this

```
while (q.size() < 1)
{
}
```

* You can use 2 loops also, O(n) only.

Q. Size of Binary Tree.

O(n) & O(H)



→ O/p : 5

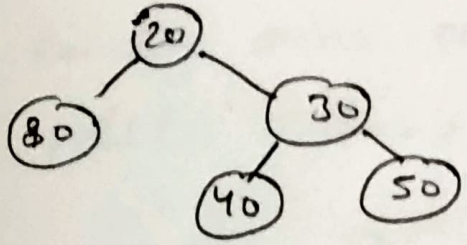
```
int getSize(Node root)
{ if(root == null)
    return 0;
  else
  & return 1 + getSize(root.left)
             + getSize(root.right);
}
```

O(H) bcz no: of active calls & height of binary tree in fn call stack

Note:

lvl order traversal can also be used to count.

Q Maximum in Binary Tree.



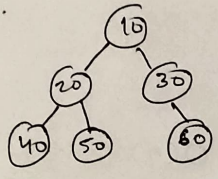
O/P : 80

O(n)
&
O(H)

```
int getMax(Node root)
{
  if (root == null)
    {return (Integer.MIN_VALUE);}
  else
    { return Math.max(root.key, Math.max(getMax(root.left),
                                         getMax(root.right)));
```

Print Left View of Binary Tree.

Q.



O/P: 10 20 40

Method 1 (Recursive) : Using Preorder traversal.

```
int max_level = 0;

void printLeft(Node root, int level)
{
    if (root == null) { return; }

    if (maxLevel < level)
    {
        System.out.println(root.key + " ");
        maxLevel = level;
    }

    printLeft(root.left, level+1);
    printLeft(root.right, level+1);
}


void printLeftView(Node root)
{
    printLeft(root, 1);
}
```

Method 2 (Iterative) : Using level order

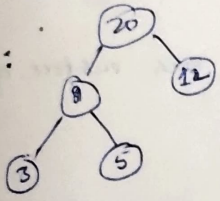
→ Exactly same except above 2 if statement and after poll() statement.

→ if (i == 0)
{ System.out.println(curr.key + " "); }

---

11. Children Sum Property.

I/P:



Sum of children (immediate). node == parent

O/P: Yes

→ boolean isSum(Node root)
{ if (root == null) return true;
if (root.left == null && root.right == null)
{ return true; }

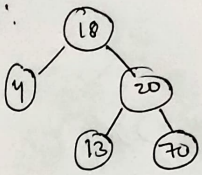
int sum = 0;
if (root.left != null) { sum -= root.left.key; }
if (root.right != null) { sum += root.right.key; }

return (root.key == sum &&
isSum(root.left) &&
isSum(root.right));

}

Q Check for Balanced Tree.

Ip:



Op: Yes.

for any node, diff b/n (height of left tree) - (height of right tree) ≤ 1.

18 → 1
4 → 0
20 → 0
} ∴ Yes.

→ Sol.

Naive

→ To call/calculate height of each subtree of every node. $O(n^2)$.

```
boolean isBalanced(Node root)
{ if(root == null) return true;
  int lh = height(root.left);
  int rh = height(root.right);
  return (Math.abs(lh - rh) <= 1 &&
          isBalanced(root.left) &&
          isBalanced(root.right));
}
```

O(n) method → Same idea, but we calculate isBal & height together.

```
int isBalanced(Node root)
{ if(root == null) return 0;
  int lh = isBalanced(root.left);
  if(lh == -1) return -1;
  int rh = isBalanced(root.right);
  if(rh == -1) return -1;
  if(Math.abs(lh - rh) > 1) return -1;
  else {return (Math.max(lh, rh) + 1;}
}
```

Check for left subtree + height.

right tree + height.