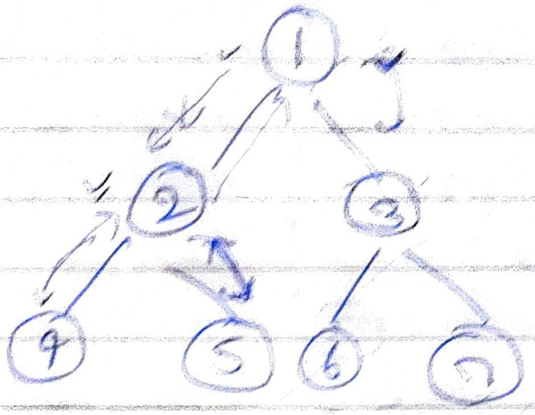
Binary Trees

1) Get sum of all nodes in a binary tree



o/p :- 28

Sol^{n} DFS or BFS

↓

SumOfNodes = 0 1 3 7 12 15 28

O(n)

↳ number of nodes

O(n)

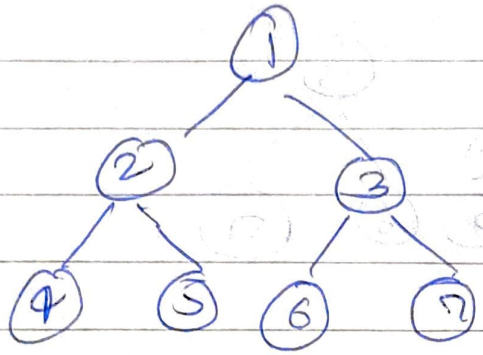
SumOfNodes = 0

```
def sumOfAllNodes(root):
    global SumOfNodes
    if root is None:
```

return

sumofNodes += root.val

sumOfAllNodes(root.left)
sumOfAllNodes(root.right)



BFS

1 | 2 | 3 | 4 | 5 | 6 | 7

total = 0 1 3 6 10 15 21 28

O(n)
O(n)

```
def sumOfAllNodes(root):
    total = 0

    if root is None:
        return total
```

queue = collections.deque([root])

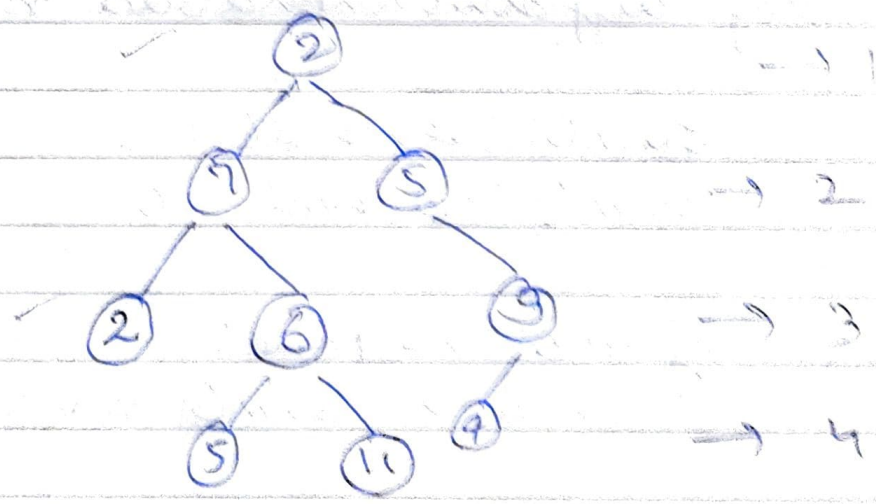
while queue:
    currentNode = queue.popleft()
    total += currentNode.val

    if currentNode.left is not None:
        queue.append(currentNode.left)

    if currentNode.right is not None:
        queue.append(currentNode.right)

return total

2) Get difference of values at even and odd levels

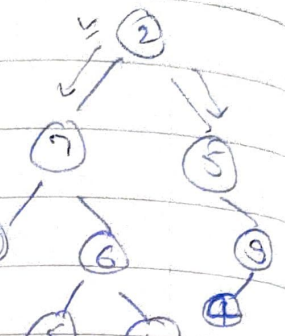


19 − 32 = -13

Need to return the difference of sum of all nodes present at

odd level) and the sum of all nodes present at even level)

BFS or DFS



(2,1) (7,2) (5,2) (2,3) (6,3)

sumOfEvenLevel = 0 2 9 11 17

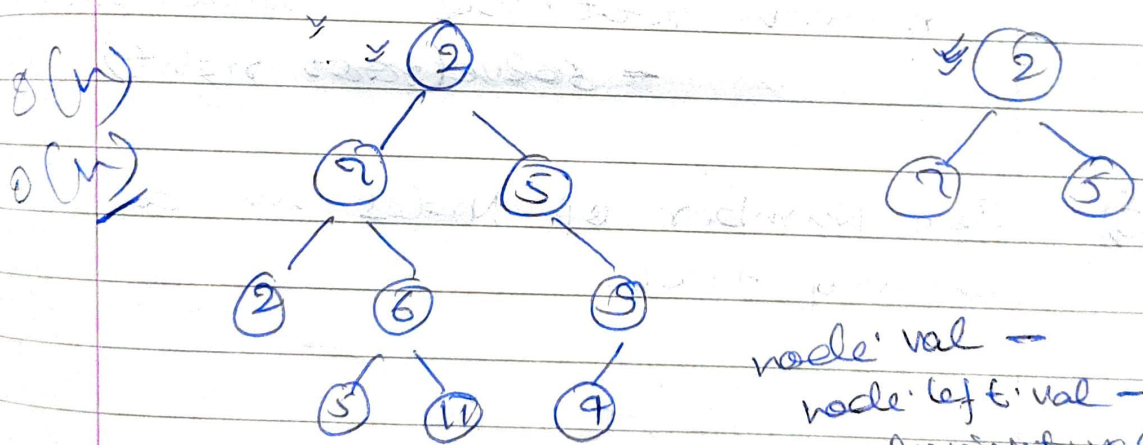
sumOfOddLevel = 0 7 12 14 23

(5,3) (6,3) (2,4) (5,4) (9,4)

```
def diffBtwEvenAndOdd(root):

    sumOfEvenLevel = 0
    sumOfOddLevel = 0

    if root is None:
        return sumOfEvenLevel

    queue = collections.deque([[root,1]])

    while queue:
        node, level = queue.popleft()
```

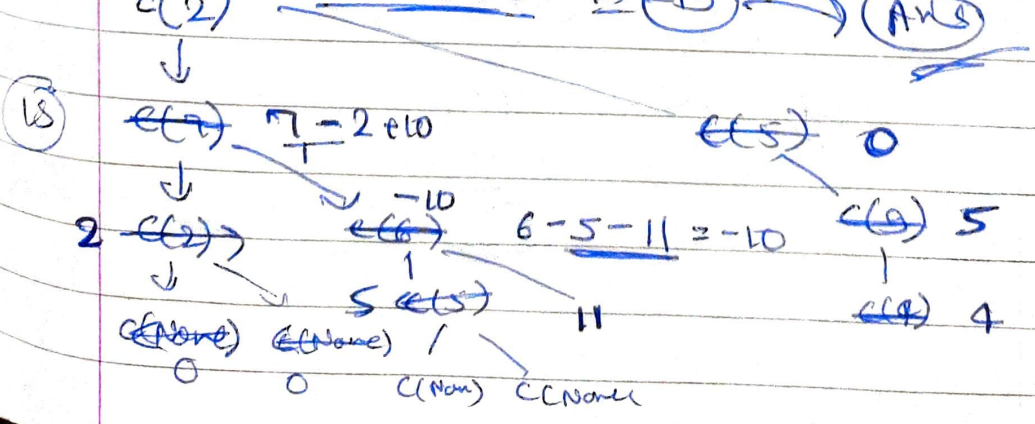
```
if level % 2 :
    sumOfOddLevel += node.val
else:
    sumOfEvenLevel -= node.val
if node.left is not None:
    queue.append((node.left, level+1))
if node.right is not None:
    queue.append((node.right, level+1))
return sumOfEvenLevel - sumOfOddLevel
```

DFS



node.val − node.left.val − node.right.val

2 − 15 + 0 = −13 → Ans

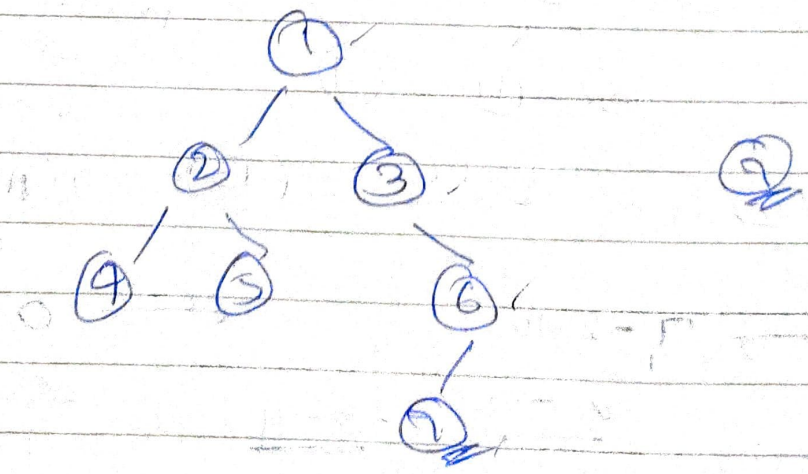


We can solve this question by performing DFS starting at the root node.

We can see that the required value is node.val - node.left.val - node.right.val and we will recursively calculate the difference using this

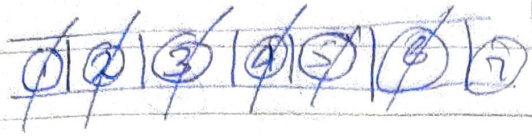
```
def solve(root):

    if root is None:
        return 0

    return root.val - solve(root.left)
                    - solve(root.right)
```

3) Get Number of Nodes in a binary tree.



Page No.:
Date:
YOUVA

Sol^n BFS or DFS

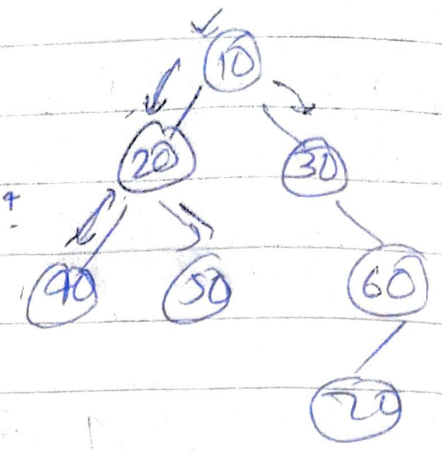


count = 0 1 2 3 4 5 6 7

O(N)
O(N)

```
def numberOfNodes(root):
    count = 0

    if root is None:
        return count

    queue = collections.deque([root])

    while queue:
        current = queue.popleft()
        count += 1

        if current.left is not None:
            queue.append(current.left)

        if current.right is not None:
            queue.append(current.right)

    return count
```

DFS



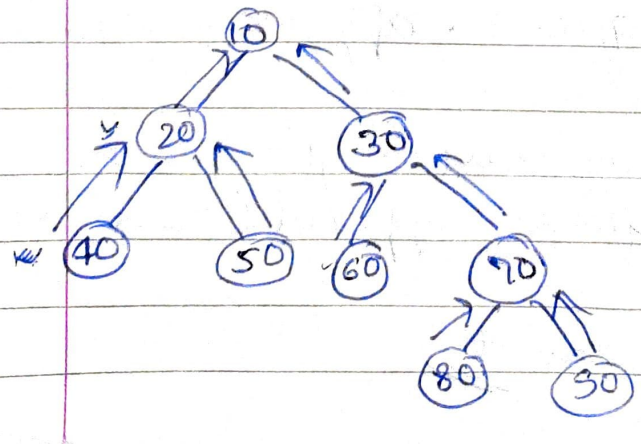
count [0] [1] [2] 3 !

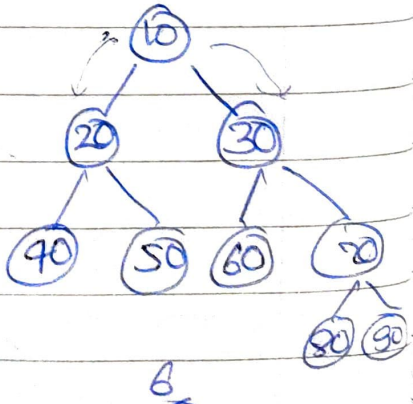
count = 0

```
def countNodes(root):
    global count
    if root is None:
        return
    count += 1
    countNodes(root.left)
    countNodes(root.right)
```

4) Get number of leaf Nodes in Binary Tree

Soln DFS or BFS





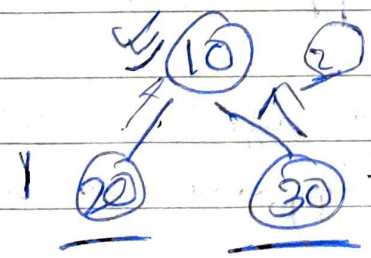
6

countLeafNodes = 6

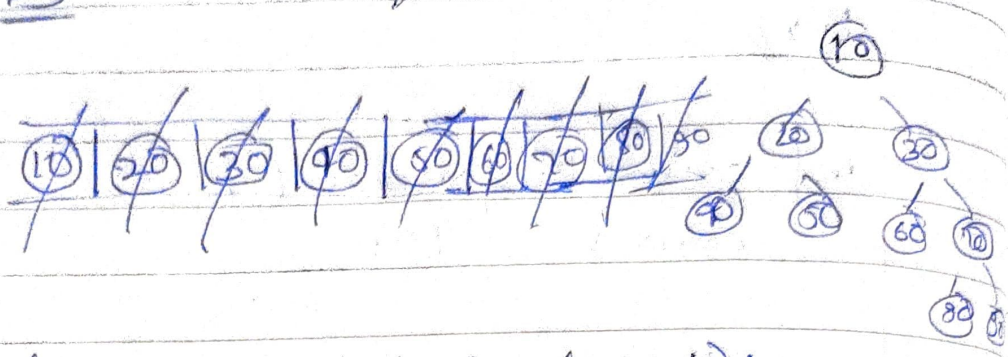
1 2 3 4 5

O(n)
O(n)

```
def countLeafNodes(root):
    count = [0]

    if root is None:
        return count[0]

    countLeafNodesUtil(root, count)
    return count[0]

def countLeafNodesUtil(root, count):

    if root.left is None and root.
        right is None:
        count[0] += 1

    if root.left is not None:
        countLeafNodesUtil(root.left, count)

    if root.right is not None:
        countLeafNodesUtil(root.right, count)
```

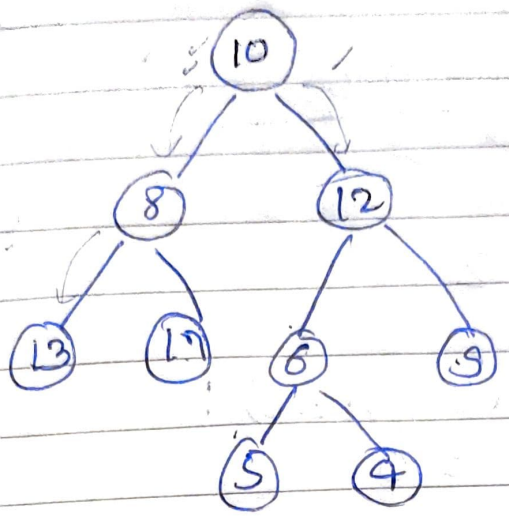


BFS count = 0 1 2 3 4 5



```
def countLeafNodes(root):
    count = 0

    if root is None:
        return count

    queue = collections.deque([root])

    while queue:
        current = queue.popleft()

        if (current.left is None and
            current.right is None):
            count += 1

        if current.left is not None:
            queue.append(current.left)

        if current.right is not None:
            queue.append(current.right)

    return count
```

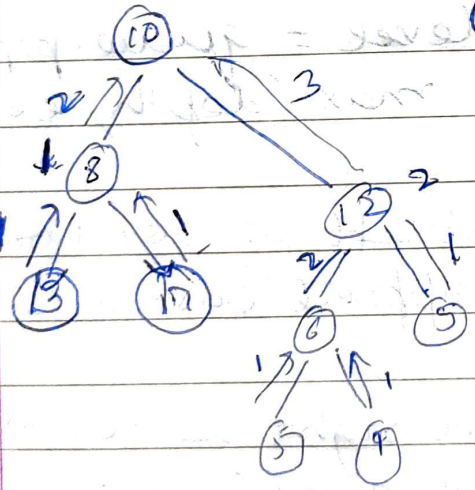
2) Get height of a binary tree.



4

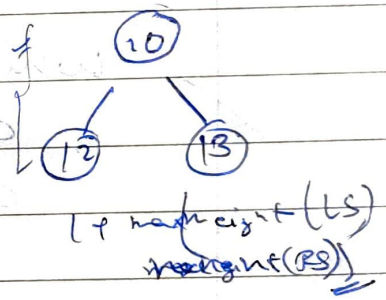
Sol^n. Max. number of nodes along the path from root to leaf node.

DFS or BFS



O(N)

O(N)

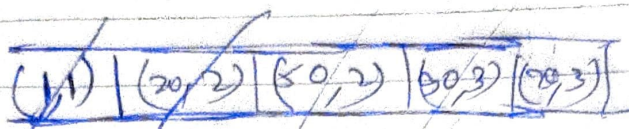


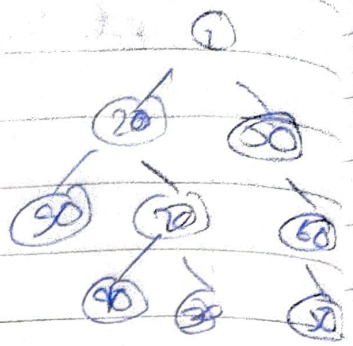
1+3 → 4

```
def height(root):
    if root is None:
        return 0
    return 1 + max(height(root.left),
                   height(root.right))
```

Page No.
Date:
youva

BFS





depth = 0 1 2 3

```
def height (root):
    depth = 0

    if root is None:
        return depth

    queue = collections.deque([root])

    while queue:
        current, level = queue.popleft()
        depth = max(depth, level)

        if current.left is not None:
            queue.append([current.left, level+1])

        if current.right is not None:
            queue.append([current.right, level+1])

    return depth
```

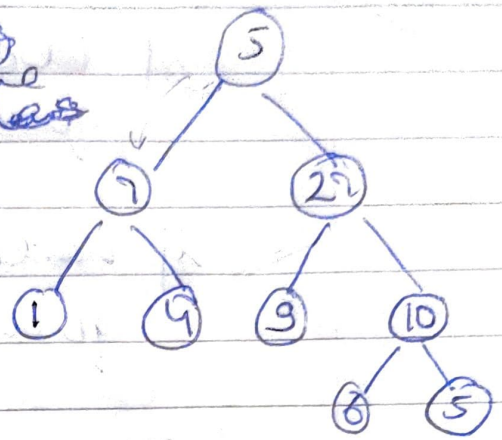
O(n)

O(n)

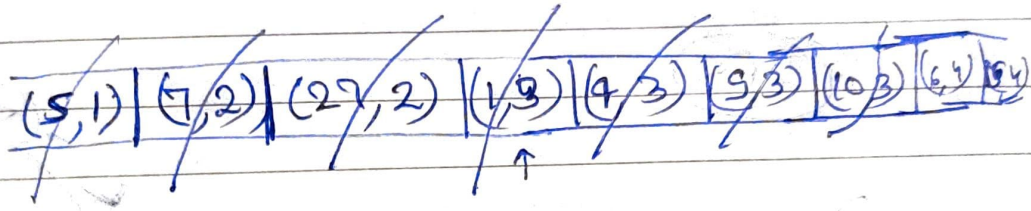
3) Print elements at given level in binary tree

level = 3 → printing the value of nodes

1, 4, 9, 10



Soln BFS (LOT)
DFS



1, 4, 9, 10

O(n)
O(n)

```
def printNodesAtGivenLevel(root, level):

    if root is None:
        return

    queue = collections.deque([(root, 1)])

    while queue:
        node, currentLevel = queue.popleft()
```

if level == currentLevel:
print(node.val)

if node.left is not None:
queue.append((node.left, currentLevel+1))

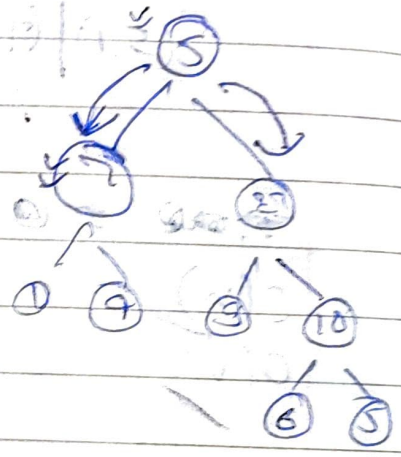
if node.right is not None:
queue.append((node.right, currentLevel+1))

return

DFS

O(n)

O(n)




```
def printNodesAtGivenLevel(root, level):
    if root is None or level <= 0:
        return

    if level == 1:
        print(root.val)

    printNodesAtGivenLevel(root.left, level-1)
    printNodesAtGivenLevel(root.right, level-1)
```

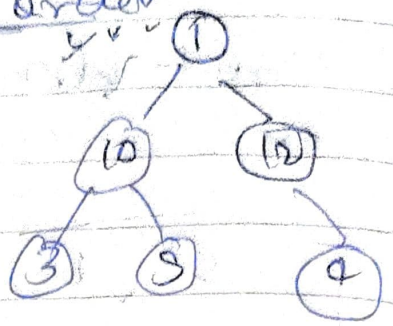
1) Print the nodes of a binary tree in level order.

2) BFS (Level order traversal)

10, 5, 6, 16, 28, 11, 13, 12, 3

```
def levelOrderTraversal(root):
    traversal = []
    if root is None:
        return traversal

    queue = collections.deque([root])

    while queue:
        currentNode, currentLevel = queue.popleft()
        traversal.append(currentNode.val)

        if currentNode.left is not None:
            queue.append((currentNode.left,
                          currentLevel+1))

        if currentNode.right is not None:
            queue.append((currentNode.right,
                          currentLevel+1))

    return traversal
```

10) Print reverse level order traversal



Search: DFS, BFS → (LOT)

(1,1) (10,2) (12,2) (3,3) (9,3) (4,3)

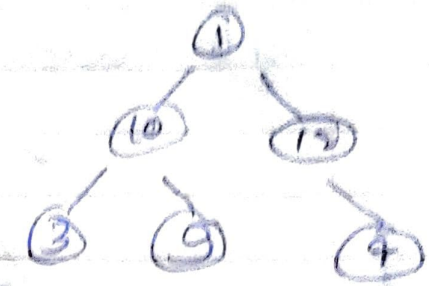
[1, 10, 12, 3, 9, 4]

[4, 9, 3, 12, 10, 1]

```
def reverseLevelOrder(root):
    traversal = []
    if root is None:
        return traversal

    queue = collections.deque([(root,1)])

    while queue:
        currentNode, currentLevel = queue.popleft()
        traversal.append(currentNode.val)

        if currentNode.left is not None:
            queue.append(currentNode.left)

        if currentNode.right is not None:
            queue.append(currentNode.right)

    return traversal
```

o/p

[3, 9, 4, 10, 12, 1]

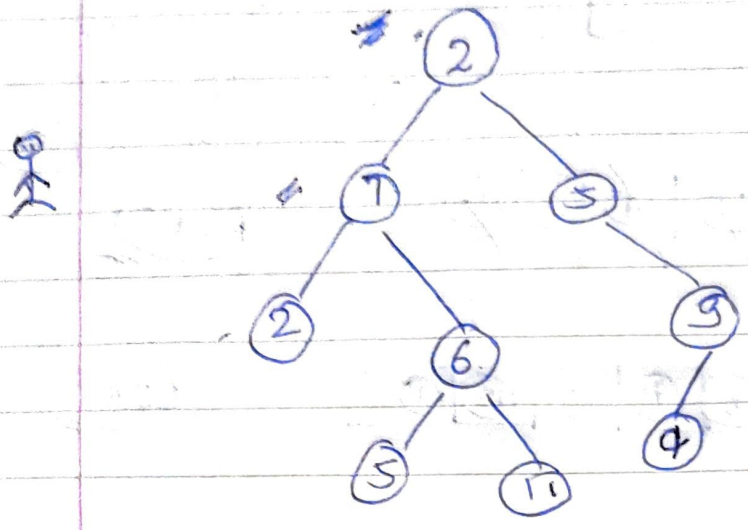


queue 1 | 12 | 10 | 4 | 9 | 3 (FIFO)

stack 3 | 9 | 4 | 10 | 12 | 1 (LIFO)

```
def reverseLevelOrder(root):

    if root is None:
        return []

    queue = collections.deque([root])
    stack = collections.deque()

    while queue:
        currentNode = queue.popleft()
        stack.appendleft(currentNode.val)

        if currentNode.right is not None:
            queue.append(currentNode.right)

        if currentNode.left is not None:
            queue.append(currentNode.left)

    return stack
```

ii) Left view of binary Tree.



o/p:- 2, 7, 2, 5

Sol^n BFS → LOT

(2,1)	(7,2)	(5,2)	(2,3)	(6,3)	(9,3)	(5,4)	(11,4)

(4,4)

depth = 0 1 2 3 4

= currentNode, currentLevel = 2, 1
7, 2
5, 2
2, 3
6, 3
9, 3
5, 4
11, 4
4, 4

BFS

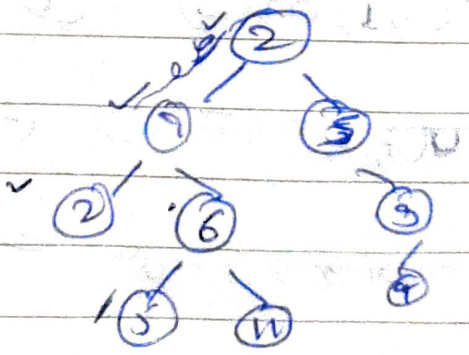
2, 7, 2, 5

O(N)

O(N)

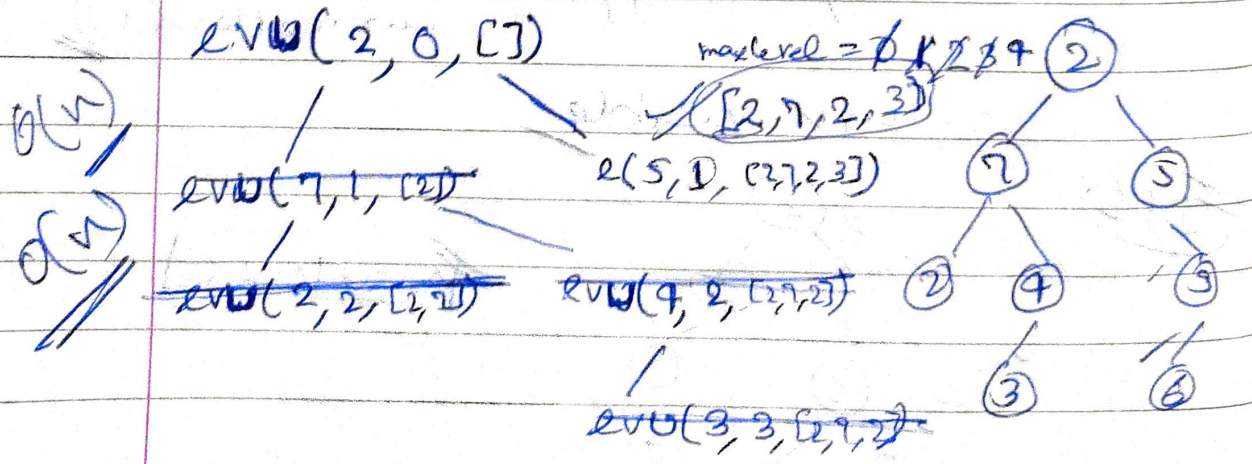
```
def leftView(root):
    visible = []

    if root is None:
        return visible

    queue = collections.deque([root])
    depth = 0

    while queue:
        currentNode, currentLevel = queue.popleft()

        if depth != currentLevel:
            visible.append(currentNode.val)
            depth = currentLevel

        if currentNode.left is not None:
            queue.append((currentNode.left,
                          currentLevel+1))

        if currentNode.right is not None:
            queue.append((currentNode.right,
                          currentLevel+1))

    return visible
```

BFS



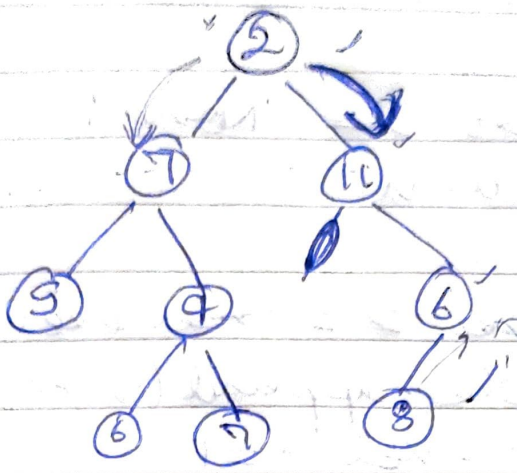
max level

```
maxlevel = 0
def leftView(root):
    visible = []

    if root is None:
        return visible

    leftViewUtil(root, 0, visible)
    return visible

def leftViewUtil(root, level, visible):
    global maxlevel
    if root is None:
        return

    if level >= maxlevel:
        visible.append(root.val)
        maxlevel += 1

    leftViewUtil(root.left, level+1, visible)
    leftViewUtil(root.right, level+1, visible)
```



Page No.
Date
youva

Print Right View of Binary Tree



// Op: [2, 11, 6, 8]

2, 11, 6, 8

maxlevel = ∅

~~1 2 3 4~~

DFS

visible

check(2, 0, [])

check(7, 1, [2, 11, 6, 8])

~~check(11, 1, [2])~~

~~check(6, 2, [2, 11])~~

~~check(None, 3, [2, 11, 6])~~

~~check(8, 3, [2, 11, 6])~~

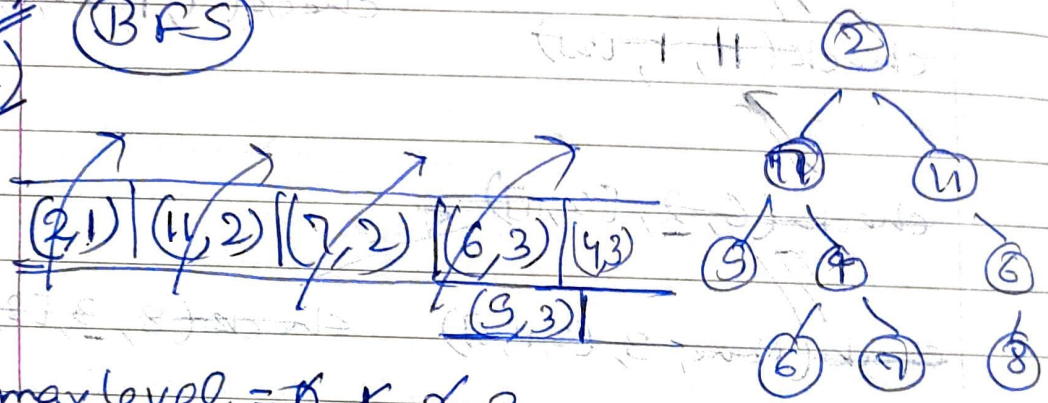
maxlevel = 0

```
def rightView(root : TreeNode):
    visible = []

    if root is None:
        return visible

    rightViewUtil(root, 0, visible)
    return visible
```

```
def rightViewUtil(root, level, visible):
    global maxlevel

    if root is None:
        return

    if level >= maxlevel:
        visible.append(root.val)
        maxlevel += 1

    rightViewUtil(root.right, level+1, visible)
    rightViewUtil(root.left, level+1, visible)
```

O(n)
O(n)

BFS



maxlevel = 0 1 2 3

2, 11, 6

(3) Prin
tree

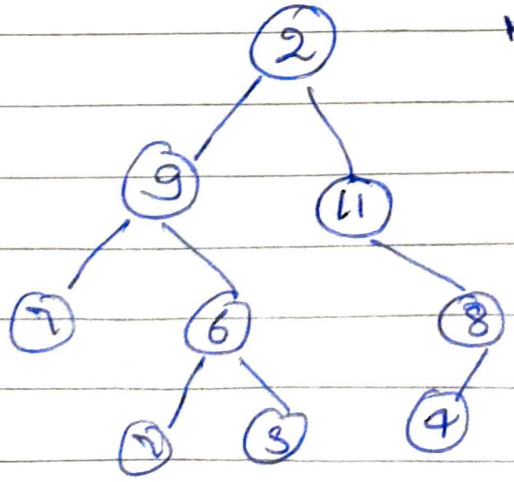
```
def rightView(root):
    visible = []

    if root is None:
        return visible
```

```
queue = collections.deque([[root,1]])
maxlevel = 0

while queue:
    currentNode, currentlevel = queue.popleft()

    if currentlevel >= maxlevel:
        visible.append(currentNode.val)
        maxlevel = currentlevel+1

    if currentNode.right is not None:
        queue.append([currentNode.right,
                      currentlevel+1])

    if currentNode.left is not None:
        queue.append([currentNode.left,
                      currentlevel+1])

return visible
```

3) Print inorder Traversal of a binary tree

↓

left Subtree
Node
Right subtree

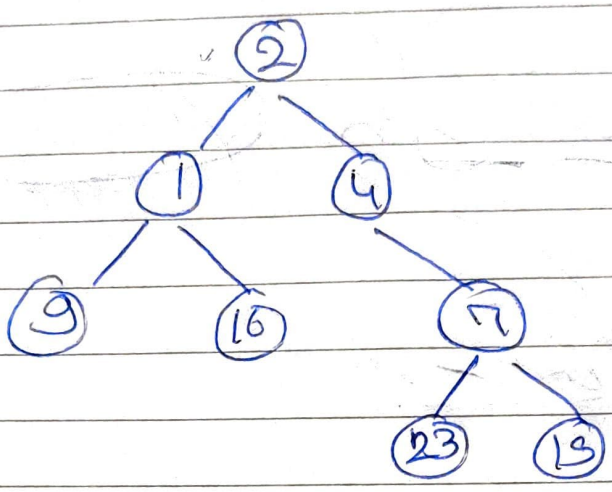


[7, 9, 1, 6, 9, 2, 11, 4, 8]

Page No.:
Date:

Binary Tree

1) Inorder Tree Traversal

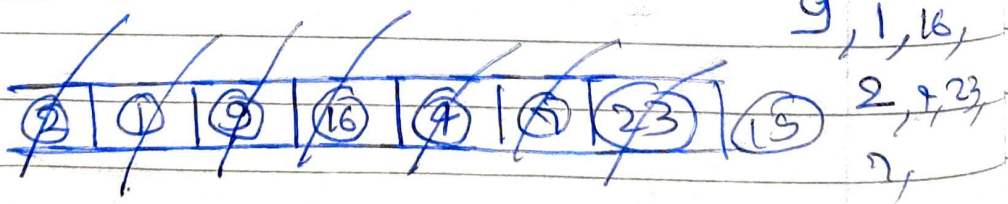


Sol: Left subtree of a Node first
Then The Node itself
Then right subtree of the node

o/p: [9, 1, 16, 2, 4, 23, 7, 13]

Recursion ✓

Iterative



9, 1, 16, 2, 4, 23, 7,

current - ~~9~~ ~~1~~ ~~16~~ ~~2~~, ~~4~~, ~~23~~, ~~7~~, ~~13~~

O(n)
O(n)

Stack → collections → Python

↓

deque

```
from collections import deque

def inorderTraversal(root):
    inorder = []

    if root is None:
        return inorder

    stack = deque()
    addNodesToStack(stack, root)

    while stack:
        current = stack.pop()
        inorder.append(current.val)

        if current.right is not None:
            addNodesToStack(stack, current.right)

    return inorder

def addNodesToStack(stack, node):
    current = node

    while current.left is not None:
        stack.append(current.left)
        current = current.left
```

O(n)
O(n)