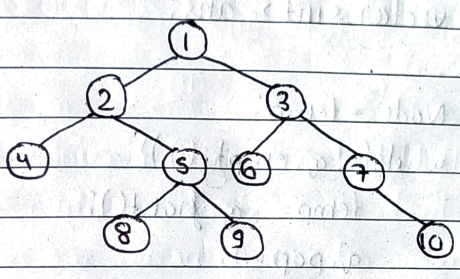
Date _____ Page _____

Day - 151

Trees - 3

* **Level Order Traversal :**



⇒ We have to traverse the tree level wise i.e.
1 2 3 4 5 6 7 8 9 10

⇒ First, we will take root and put it in a line.

⇒ Now, we will print 1 then put their left and right in the line, and pop 1 from line.

⇒ After that, we will do this same procedure again and again.

⇒ So, for line, we use queue DS.

⇒ Queue is of <Node *> type.

Code

Date _____ Page _____

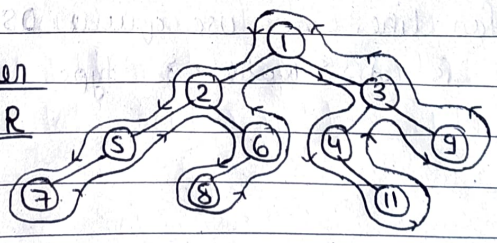
```
vector<int> levelOrder(Node *root){
    queue<Node *> q;
    q.push(root);
    ~~vector.push_back~~
    vector<int> ans;

    Node *temp;
    while(!q.empty()){
        temp = q.front();
        q.pop();
        ans.push_back(temp->data);
        if(temp->left)
            q.push(temp->left);
        if(temp->right)
            q.push(temp->right);
    }
    return ans;
}
```

T.C. → O(n) , S.C. → O(n)

* Shortcut trick to calculate —

Preorder
N L R



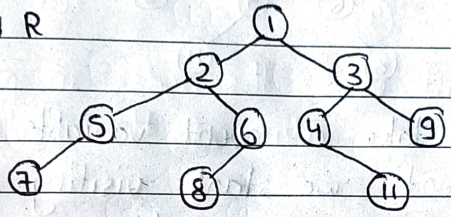
⇒ 1 2 5 7 6 8 3 4 11 9

Date: Page:

- We have to first print Node.
- So, whenever first time we get any node, just print it.

Inorder

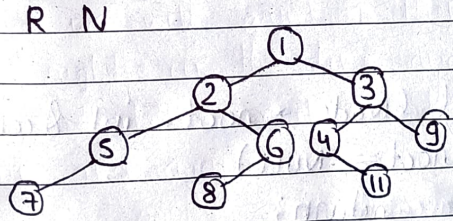
L N R



- When we get any node second time, then we will print it.
- 7 5 2 8 6 1 4 11 3 9

Postorder

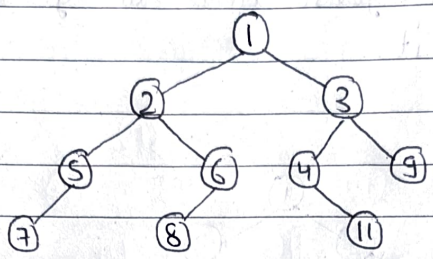
L R N



- When we get any node third time, then we will print it.
- 7 5 8 6 2 11 4 9 3 1
- So, if we are visiting any node last time then print it.

Date _____ Page _____

* Size of Binary Tree:



⇉ we will take a count variable.

⇉ After that, we start visiting our tree.

⇉ When we get any node, we will increase the count by one.

⇉ Another way to understand this is as we get any node then we will increase the count by 1 & then goto left side and after that right side.

Code

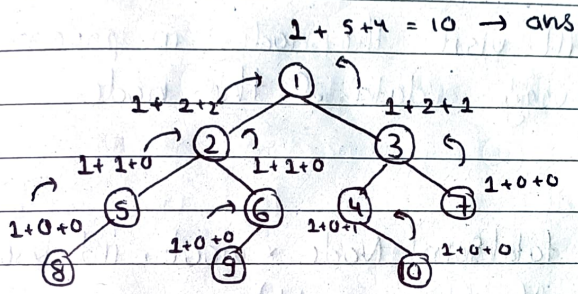
```
void total (Node * root, int &count){
    if(root == NULL)
        return;
    count++;
    total( root->left, count);
    total( root->right, count);
}
```

Date _____ Page _____

Another method:

⇒ When we get any node, then we will ask it to return no. of left side node and right side node.

⇒ Then we will plus 1 into it & return the ans.



⇒ So,

1 + total node in left side + Total node in right side.

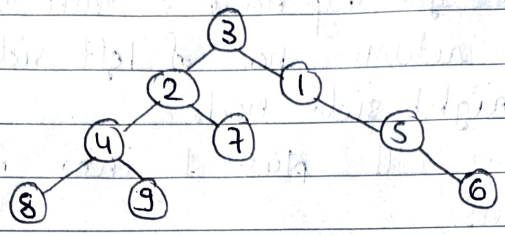
Code

```
int totalNode(Node *root){
    if(root == NULL)
        return 0;
    return (1 + TotalNode(root->left) +
            totalNode(root->right);
}
```

T.C. → O(n) , S.C → O(n)

Date _____ Page _____

* Sum of Binary Tree:



⇒ we will visit the nodes in preorder and sum every data of the node.

Code

```
void totalSum( Node * root, int &sum){
    if(root == NULL)
        return;
    sum += root → data;
    totalSum( root → left, sum);
    total sum( root → right, sum);
}
```

Another approach

⇒ we will calculate ——
root → data + left side sum + right side sum

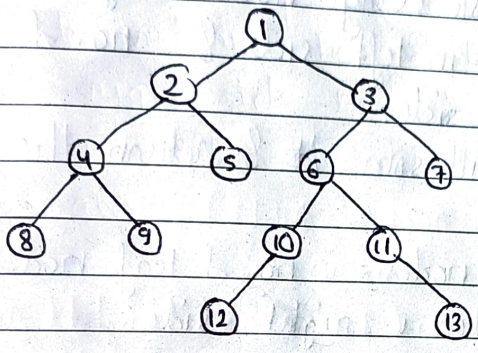
Code

```
int totalsum( Node *& root){
    if( root == NULL)
        return 0;
    return ( root → data + totalsum(root→left)
    + totalSum( root → right));
}
```

Date

Page

* Count Leaves in BT:



⇉ Leaves are that nodes that don't have any children.

⇉ We have to return the no of leaf nodes.

⇉ We will traverse the tree and ask every node that is leaf node or not.

⇉ If yes then increase the count otherwise goto left & right side.

Code

```
void countLeaf( Node * root, int & count){
    if(root == NULL)
        return;
    if( !root->left && !root->right ){
        count++;
        return;
    countLeaf( root->left, count);
    countLeaf( root->right, count);
}
```

Date _____ Page _____

Another method

⇉ we will ask every node to give no. of leaf nodes in left side & no. of leaf nodes in right side.

⇉ Then we will sum it & return the sum.

So,

No. of leaf nodes = no. of leaf nodes in left side + right side;

⇉ when any node is leaf node.. then it will return 1.

Code

```
int countLeaf (Node * root){
    if(root == NULL)
        return 0;
    if(!root->left && !root->right)
        return 1;
    return(countLeaf(root->left) +
        countLeaf(root->right));
}
```

* Count Non leaf Nodes:

⇉ we have to return non-leaf nodes.

⇉ we will ask every node to give left & right side non leaf nodes.

Date _____ Page _____

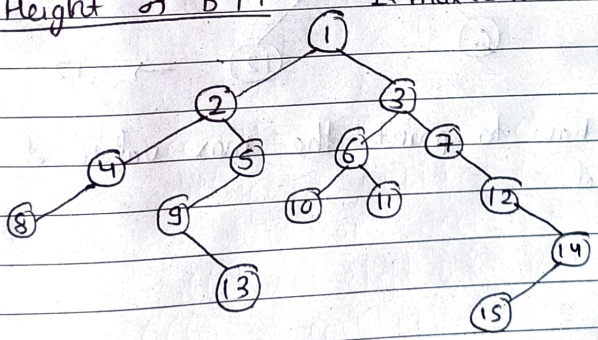
⇒ So, 1 + no. of non-leaf nodes in left side + right side.

⇒ When any node is leaf node then it will return 0.

Code

```
int nonLeaf( Node *root){
    if(root == NULL)
        return 0;
    if(!root->left && !root->right)
        return 0;
    return( 1 + nonLeaf(root->left) +
        nonLeaf(root->right));
}
```

* Height of BT: 1+max(L,R)



⇒ We will traverse & count nodes ~~in~~ from ~~every root~~ root to every leaf nodes.

⇒ ~~which~~ If it is greater than previous height then update it.

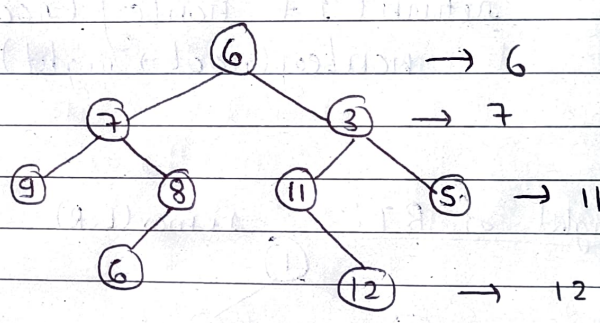
Date Page

⇉ So, 1 + max (Height of left side , Height of right side).

Code

```
int height( Node * root){
    if( root == NULL)
        return 0;
    return ( 1 + max( height( root→left),
        height( root→right ));
}
```

* Largest value in each level:



⇉ We have to print the max value of each level.

⇉ To solve this problem, first, we will push the root element in the queue.

⇉ Now, we will take a count variable.

⇉ And store 1 into it.

⇉ Count variable is used to count the no. of elements in that level.

Date

Page

=) Now, we will find the max elements upto that count value.

=) After that, we will push the max element into ans vector.

=) Now, we will store the left & right of every node of that level in the queue.

=) And maintain a count_temp variable to know that no. of nodes in the next level.

=) Now, we will do the same process again.

=) And in the end, we will return the ans vector.