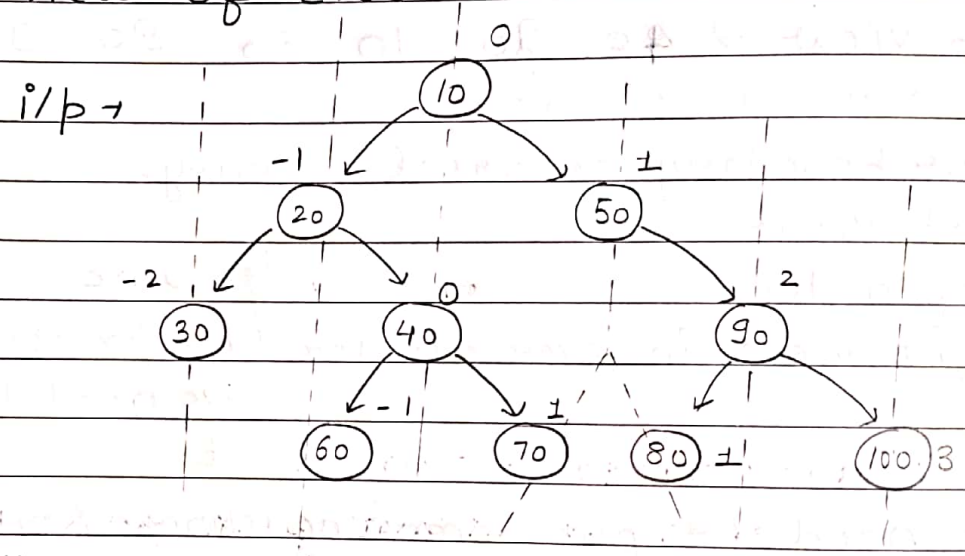
13/05/2023

Q1 Top view of tree



o/p → 30 20 10 50 90 100

The above question can be done with the help of horizontal distance from the root node. If we go towards left, reduce by 1 & if we go towards right then increment by 1.

If for a particular answer is stored already, then don't store else store the node. Here the traversal that we have to follow is level order traversal as in this higher level is processed first

Stored	Not stored
0 → 10	40
-1 → 20	60
1 → 50	70, 80
-2 → 30	-
2 → 90	-
3 → 100	-

Code

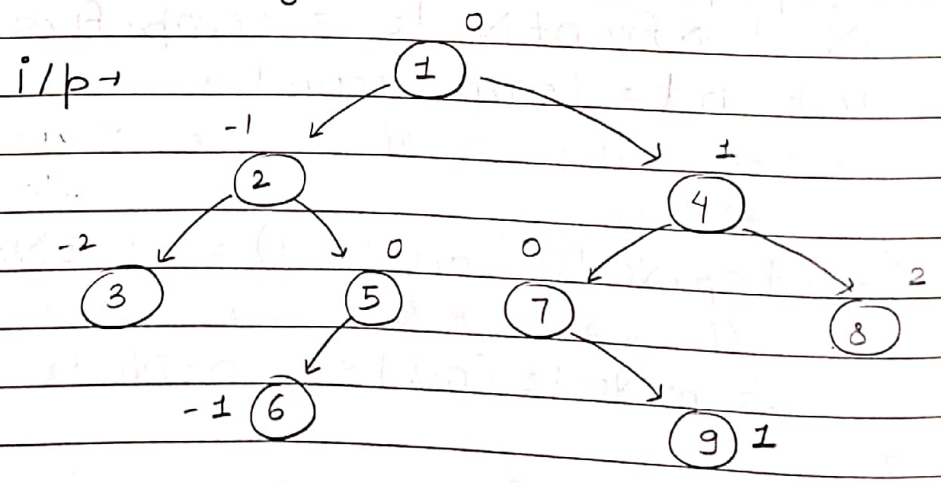
```
void printTopView (Node * root){
    // Tree empty
    if (root == NULL){
        return;
    }
    //Map which stores horizontal distance &
    node data
    map <int, int>  topNode;
    //Level order traversal
    //Create queue consisting of pair having
    Node *  & horizontal distance
    queue <pair <Node *, int >> q;
    //Push root & hd of root
    q.push (make_pair (root, 0));
    while (!q.empty ()){
        pair <Node *, int> temp = q.front();
        q.pop();
        Node* frontNode = temp.first;
        int hd = temp.second;
        //Check whether any node is already
        present for hd.
        if (topNode.find (hd) == topNode.end()){
            //Not present & hence store
            topNode [hd] = frontNode -> data;
        }
        //Push left child & also hd
        if (frontNode -> left){
            q.push (make_pair (frontNode -> left, hd-1));
        }
        //Push right child & also hd
```

```
        if (frontNode -> right) {
            q.push (make_pair (frontNode -> right, hd+1));
        }
    }
    // Top view stored in the map topNode
    for (auto i : topNode) {
        cout << i.second << endl;
    }
}
```

Note → topNode.find (hd) == topNode.end () ⇒
While searching for hd, we have reached the end and this means hd was not found.

topNode.find (hd) != topNode.end () ⇒
This means entry was found i.e hd is present in the map.

Q2 Bottom view of the tree



O/p → 3 6 7 9 8
↳ not 5 as given in question

Again we will be using the concept of horizontal distance from root node.

0 → ~~1~~ ~~5~~ 7
-1 → ~~2~~ 6
1 → ~~4~~ 9
-2 → 3
2 → 8

Here we have to the updations in map even if the entry is present in the map.

-2	-1	0	1	2	
↓	↓	↓	↓	↓	
3	6	7	9	8	} Ans

Note → What if question said to store 5 instead of 7? In this case we would be storing the hd and vector and as we go on to next level remove previous answer.

Level-0 ⇒ 0 → {1}
Level-1 ⇒ -1 → {2}, 1 → {4}
Level-2 ⇒ -2 → {3}, 0 → {5, 7}, 2 → {8}
Level-3 ⇒ -1 → {6}, 1 → {9}

1st value will be considered.

Code
Instead of

```
if (topNode.find(hd) == topNode.end()){
        topNode[hd] = frontNode->data;
}
```

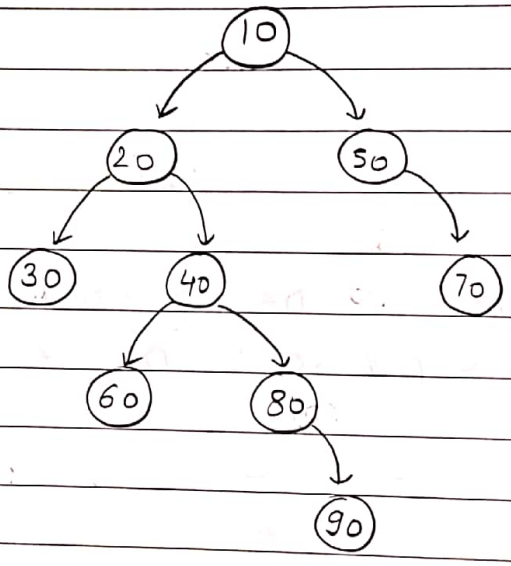
⇓

```
topNode[hd] = frontNode->data;
```

Here no if condition is used.

Q3 Left view of tree

i/p →



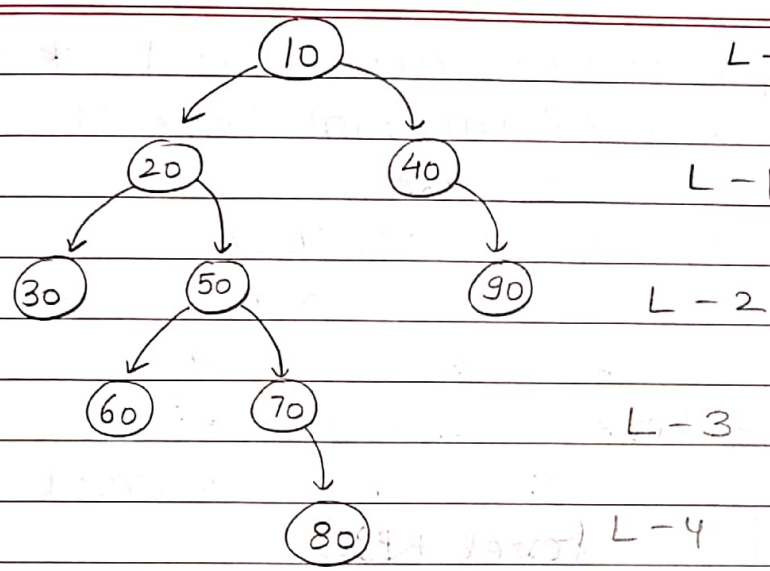
o/p → 10 20 30 60 90

This question is very simple when level order traversal is used but here we will be solving it with the help of recursion. Here we will be storing the answer only when for that particular level, no node is present. If node is present for that level, then do not update it

	Not stored
0 → 10	-
1 → 20	50
2 → 30	40, 70
3 → 60	80
4 → 90	-

Ans → 10, 20, 30, 60, 90

The above method is also simple. Lets do it with recursion.



Level → 0 , vector size = 0] matching
vector → {10}
Level - 1 , vector size = 1] matching
vector → {10, 20}
Level - 2, vector size = 2] matching
vector → {10, 20, 30}
Level - 2 , vector size = 3] not matching
vector → {10, 20, 30}
Level - 3 , vector size = 3] matching
vector → {10, 20, 30, 60}
Level - 3, vector size = 4] not matching
vector → {10, 20, 30, 60}
Level - 4 , vector size = 4] matching
vector → {10, 20, 30, 60, 80}
Level - 1 , vector size = 5] not matching
vector → {10, 20, 30, 60, 80}
Level - 2 , vector size = 5] not matching
vector → {10, 20, 30, 60, 80}

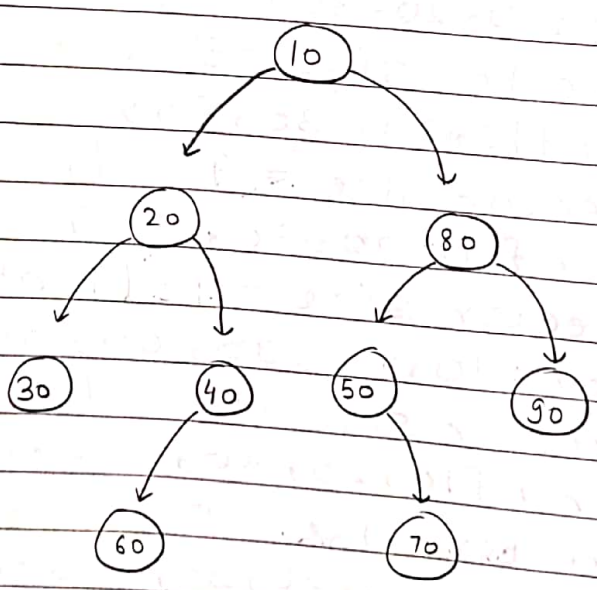
If level and size of vector is matching, then only insert element in the vector.

Code

```
void printLeftView(Node * root,
vector<int> & ans, int level){
        // Base case
        if(root == NULL)
                return;
        // level & size are same, then insert
        if(level == ans.size()){
                ans.push_back(root->data);
        }
        // left subtree
        printLeftView(root->left, ans, level+1);
        // Right subtree
        printLeftView(root->right, ans, level+1);
}
```

Q4 Right view of tree.

i/p →



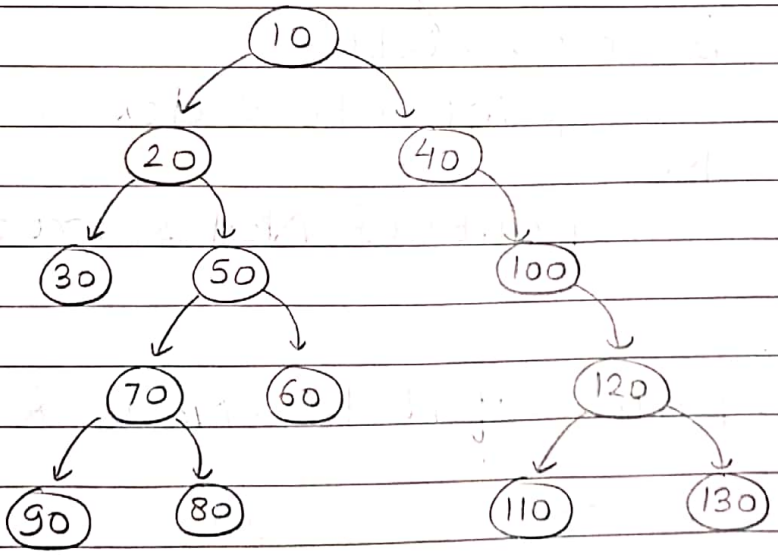
o/p → 10 80 90 70

Here first we need to go to the right & then left.

Code

```
void printRightView (Node * root, vector <int>&
ans, int level) {
        // Base case
        if (root == NULL)
                return ;
        if (level == ans.size()) {
                // Insert in vector
                ans.push_back (root -> data) ;
        }
        // Right subtree
        printRightView (root -> right, ans, level +1) ;
        // Left subtree
        printRightView (root -> left, ans, level +1) ;
}
```

Q5 Boundary traversal.



i/p →

o/p → 10 20 30 90 80 60 110 130 120 100 40

1) Print left nodes first.
2) Print leaf nodes now
3) Print right nodes in reverse order.

* Kind of pre-order traversal to print the left nodes. Stop when leaf node is found.

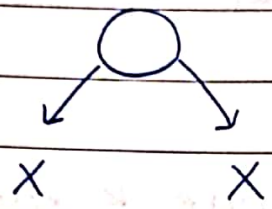
* Print leaf nodes
* Print right nodes while returning from the recursive call.

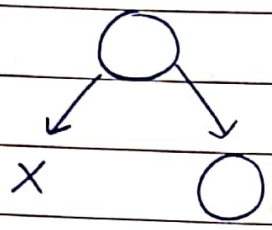
Code

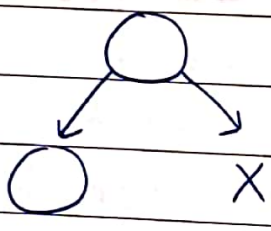
```
void printLeftNodes (Node * root){
    // Base case
    if (root == NULL)
        return;
Leaf node ← if (root→left == NULL && root→right == NULL)
        return;
    // Print the data of current node
    cout << root→data << " ";
    // Left call first
    if (root→left)
        printLeftNodes (root→left);
    else // Right call
        printLeftNodes (root→right);
}

void printLeafNodes (Node * root) {
    // Base case
    if (root == NULL) {
        return;
    }
    // Leaf node found & hence print
    if (root→left == NULL && root→right == NULL)
        cout << root→data << " ";
```

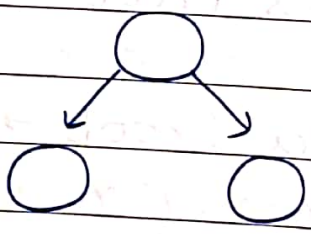
```
    printLeafNodes (root->left); //Left call
    printLeafNodes (root->right); //Right call
}

void printRightNodes (Node * root) {
    // Base case
    if (root == NULL)
        return ;
    // Leaf node found
    if (root->left == NULL && root->right == NULL)
        return ;
    // Right call first
    if (root->right)
        printRightNodes (root->right);
    else //Left call
        printRightNodes (root->left);
    // While returning print data of root
    cout << root->data << " ";
}

void boundaryTraversal (Node * root) {
    //Empty tree
    if (root == NULL)
        return ;
    // To avoid duplicacy
    cout << root->data << " ";
    // Left nodes
    printLeftNodes (root->left);
    //Leaf nodes
    printLeafNodes (root);
    //Right node
    printRightNodes (root->right);
```

Left Nodes

1)  (Return)

2)  (go to right)

3)  (go to left)

4)  (go to left)

That's why if else has been used.

Time complexity = O(n)