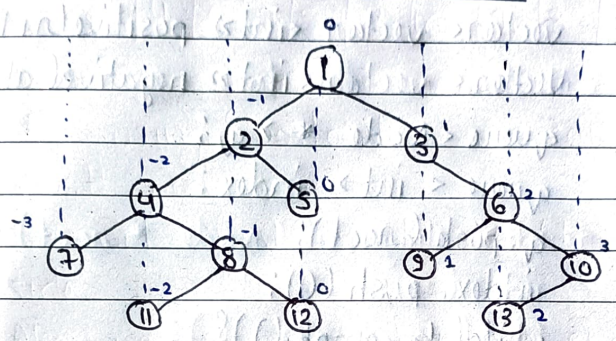
Date _____ Page _____

Day - 158

Trees - 8

* Vertical Traversal of BT:



⇒ We have to print the nodes vertically.

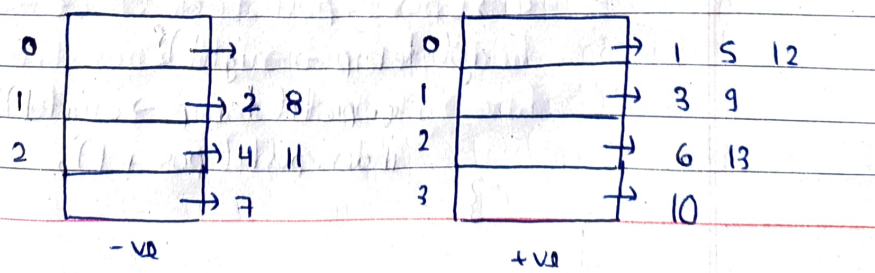
⇒ 7 4 11 2 8 1 5 12 3 9 6 13 10

⇒ So, first we print that node that have least index.

⇒ And in the same index, first we print low level node to high level node.

⇒ We can create ~~a~~ two 2D array that will store the nodes.

⇒ One array will store -ve indices nodes & other will +ve indices node.



Date _____ Page _____

Code

```
vector<int> verticalOrder(Node *root){
    int l=0, r=0;
    find(root, 0, l, r);
    vector<vector<int>> positive(r+1);
    vector<vector<int>> negative(abs(l+1));
    queue<Node *> q;
    queue<int> index;
    q.push(root);
    index.push(0);
    while(!q.empty()){
        Node *temp = q.front();
        q.pop();
        int pos = index.front();
        index.pop();
        if(pos >= 0)
            positive[pos].push_back(temp->data);
        else
            negative[abs(pos)].push_back
                (temp->data);
        if(temp->left){
            q.push(temp->left);
            index.push(pos-1);
        }
        if(temp->right){
            q.push(temp->right);
            index.push(pos+1);
        }
    }
```

Date _____ Page _____

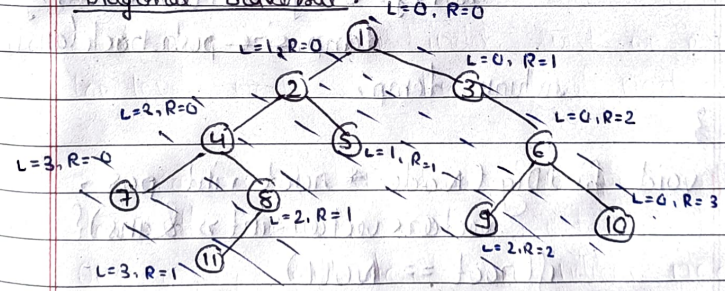
```
vector<int> ans;
for(int i= negative.size(); i>0; i--)
    for(int j= 0; j< negative[i].size(); j++){
        ans.push_back(negative[i][j]);
    }

for(int i=0; i< positive.size(); i++)
    for(int j= 0; j< positive[i].size(); j++)
        ans.push_back(positive[i][j]);

return ans;
}
```

T.C. → O(n) S.C → O(n)

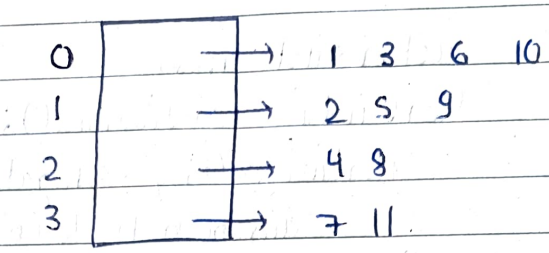
* Diagonal Traversal :



⇒ We have to print elements diagonally i.e.
1 3 6 10 2 5 9 4 8 7 11

⇒ So, we can create a 2D array that will store diagonal elements of that ~~dig~~ no. of diagonal.

Date Page



⇒ We will traverse the tree as N L R.

Code

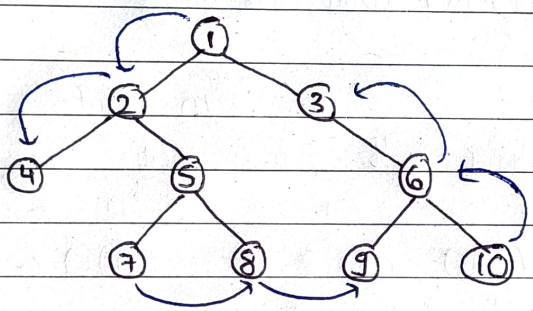
```
vector<int> diagonal(Node *root){
    int l=0;
    find(root, 0, l);
    vector<vector<int>> ans(l+1);
    findDig(root, 0, ans);
    vector<int> temp;
    for(i=0; i<ans.size(); i++
        for(j=0; j<ans[i].size(); j++)
            temp.~~size~~ push_back(ans[i][j])
    return temp;
}

void findDig(Node *root, int pos,
             vector<vector<int>> &ans){
    if(root == NULL)
        return;
    ans[pos].push_back(root->~~ans~~ data);
    findDig(root->left, pos+1, ans);
    findDig(root->right, pos, ans);
}
```

Date _____ Page _____

```
void find( Node *root, int pos, int &l){
    if(!root) return;
    l = max(pos, l);
    find(root->left, pos+1, l);
    find(root->right, pos, l);
}
```

* **Boundary Traversal :**



⇒ We have to print the elements that are on the boundary. i.e. —

1 2 4 7 8 9 10 6 3

⇒ So, here, first we have traverse the left side boundary then leaf nodes & then right side boundary in reverse order.