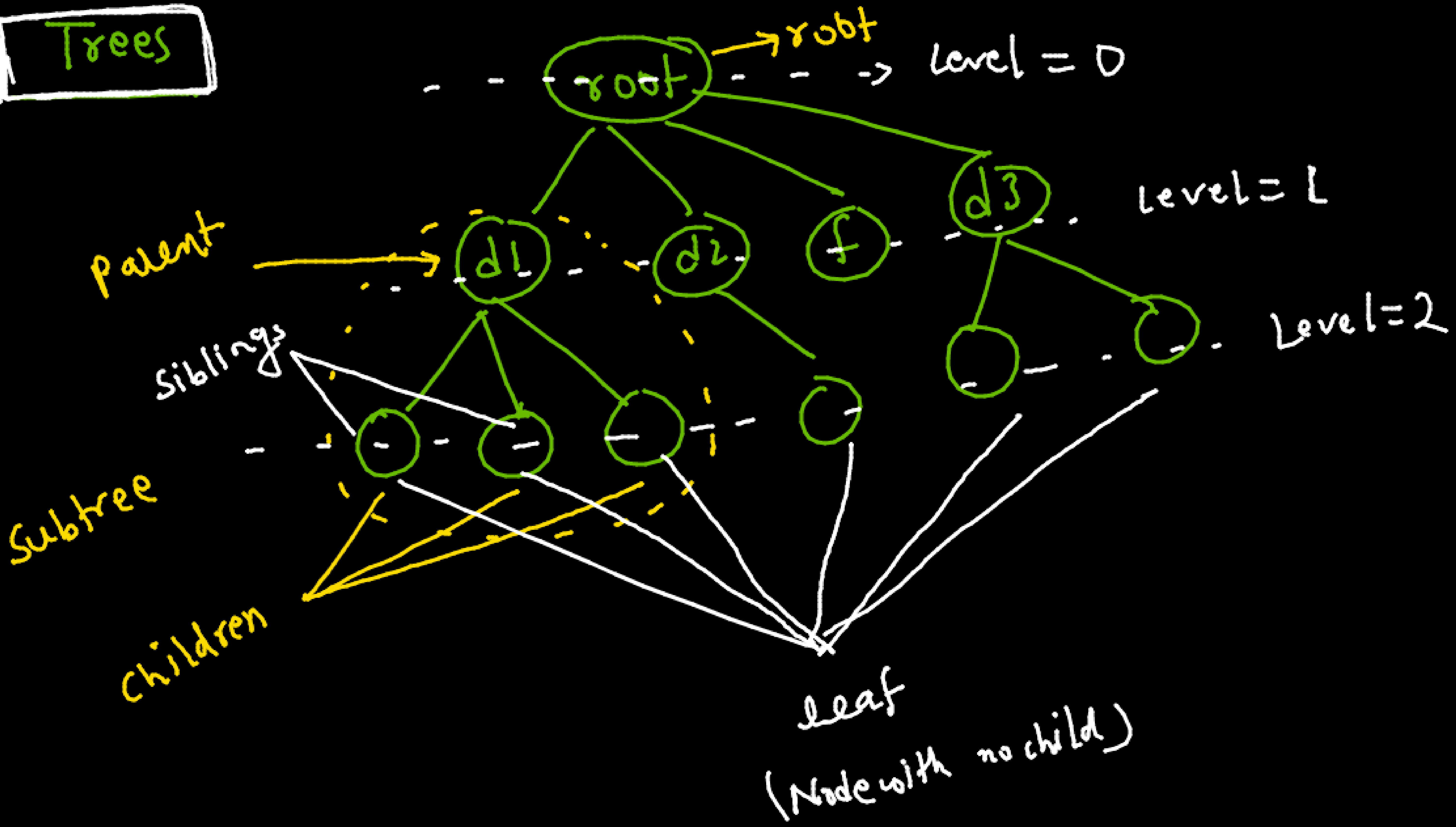
Trees

Important Question

Pre, Post Inorder Traversal

1. Print level wise (recursion)(queue)
2. count Nodes (recursion)
3. Find height (recursion)
4. count leaf node (recursion)(queue)
5. second largest (queue)
6. Replace with depth (recursion)

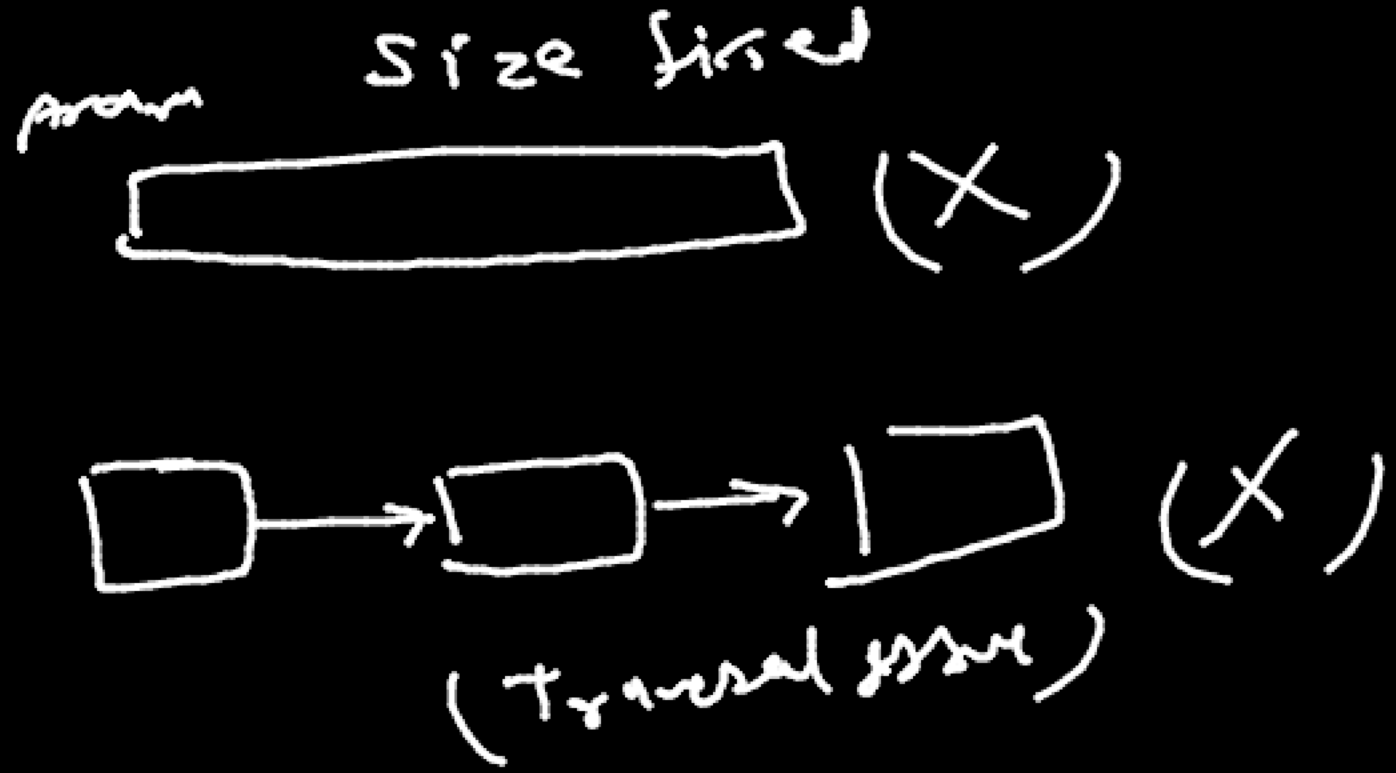
Trees



root
root
level = 0
d1
d2
f
d3
level = 1
Level = 2
Parent
Siblings
Subtree
Children
leaf
(Node with no child)

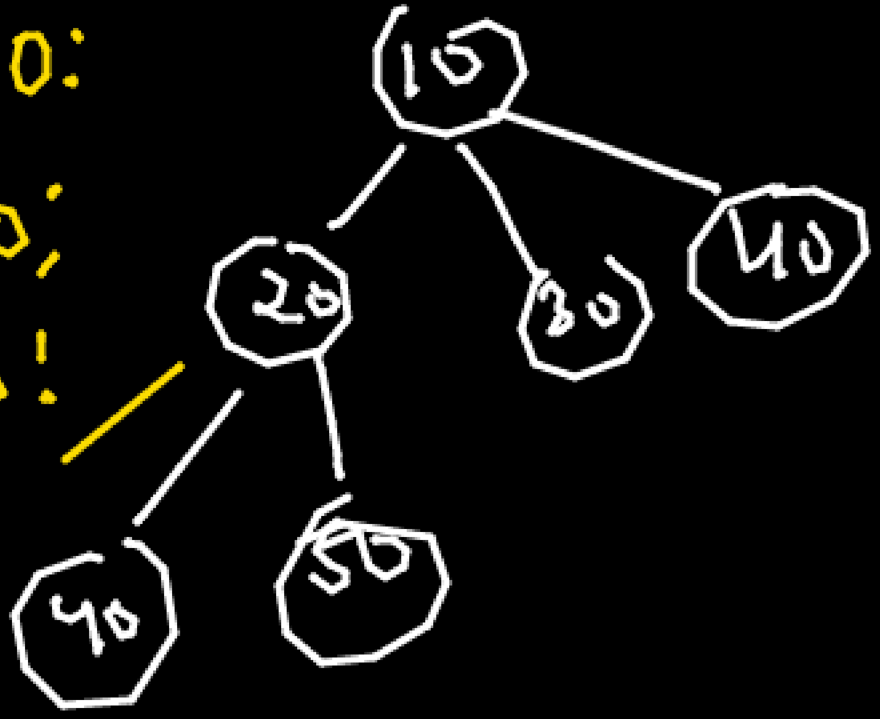
Generic Tree class

```
class TreeNode {
public:
    T data;

    vector<TreeNode<T>*> children;

    TreeNode(T data) {
        this->data = data;

};
```



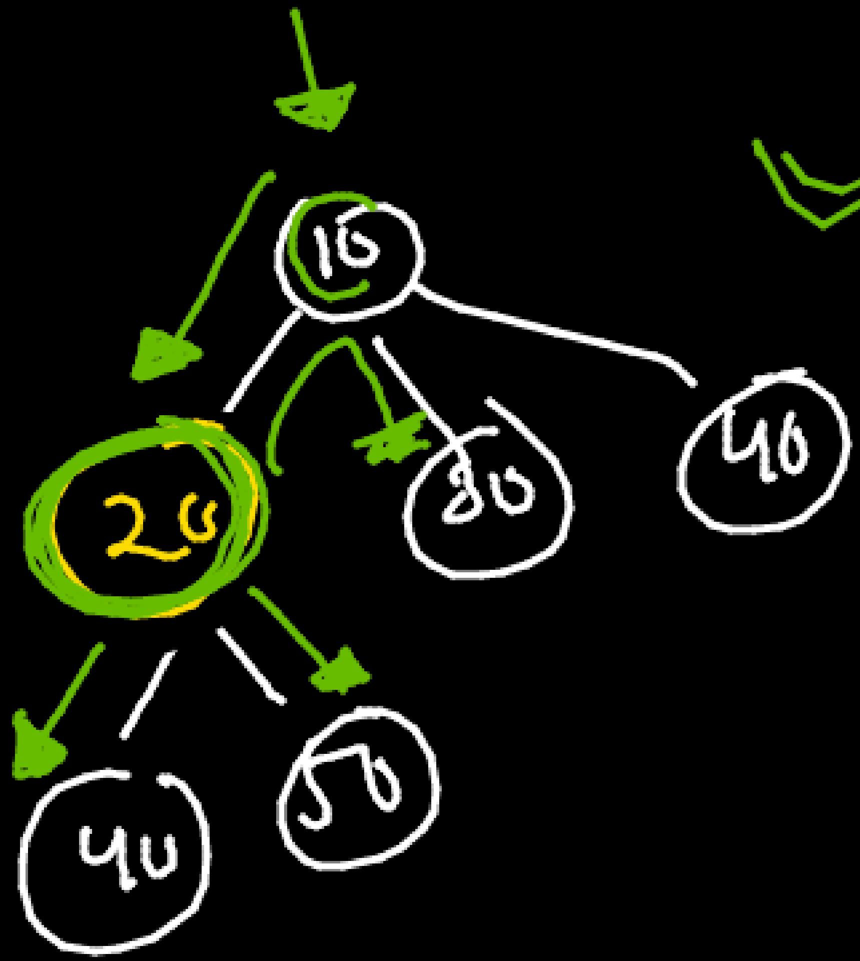
10 : 20, 30, 40,   10 :
20 : 40, 50,
30 :
40 :
40 ;
50 !

Print level Wise



```
void printlevel (TreeNode <int>* root) {
    if (root == NULL) return;

    queue <TreeNode <int>*> pendingNodes;
    pendingNode.push (root);

    while (pendingNode.size() != 0) {
        TreeNode<int> *first = pendingNode.front();
        pendingNode.pop();
        cout << first ->data << ":";
    for (int i=0; i< first -> children.size() ;i++){
            cout << first ->children[i].data <<" ";
        pendingNode. push (first -> children[i] ;

    }

    cout << endl;

  }
```

Sum of all Nodes



```
int fun ( TreeNode<int>* root ) {  → if(root==NULL)
                                        return 0;
    int sum = root -> data;

    for (int i=0; i<root -> children.size(); i++)
        sum += fun (root -> children[i]);

    return sum;

}
```

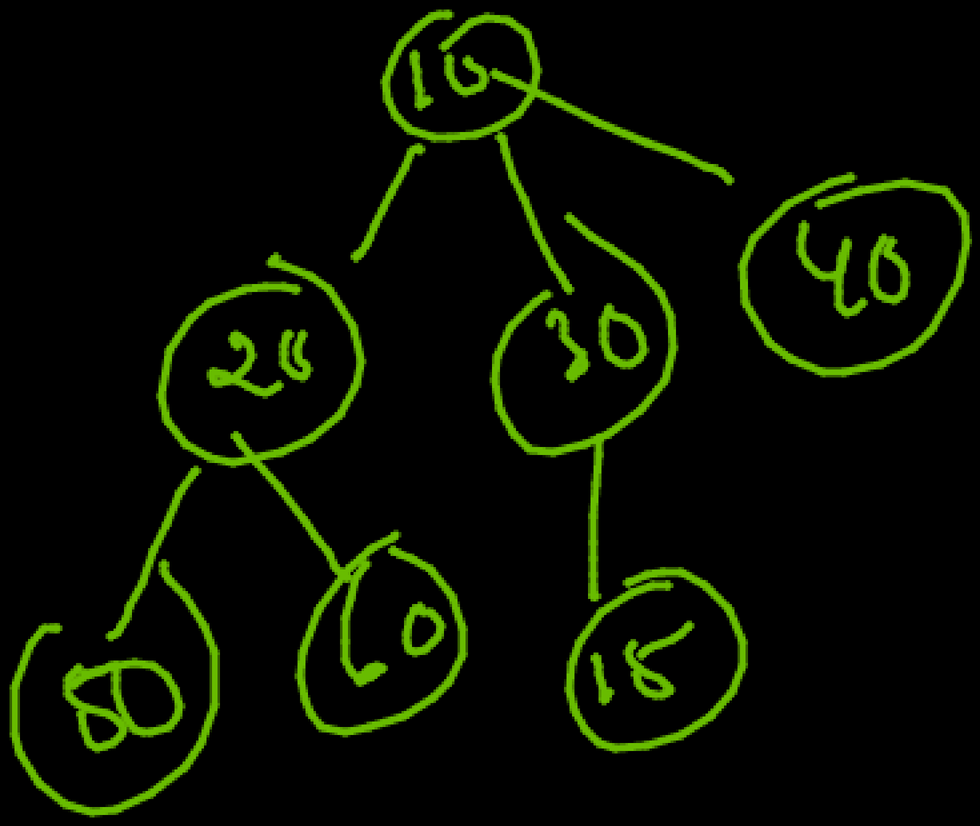
sum

10 +20 +40 +50

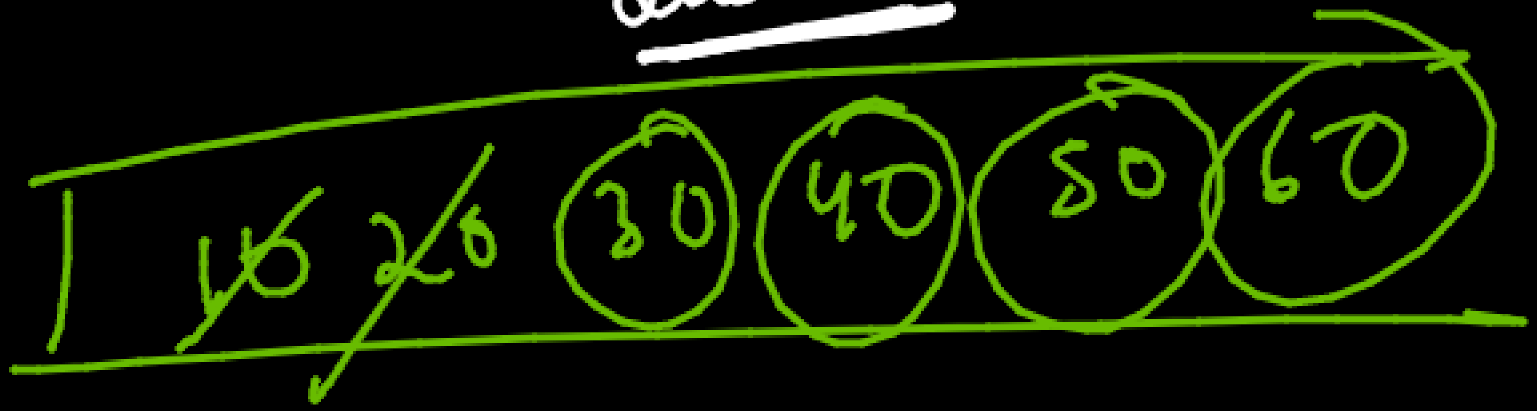
30 +40

MAX DATA NODE

→ easily can be done by Queue.



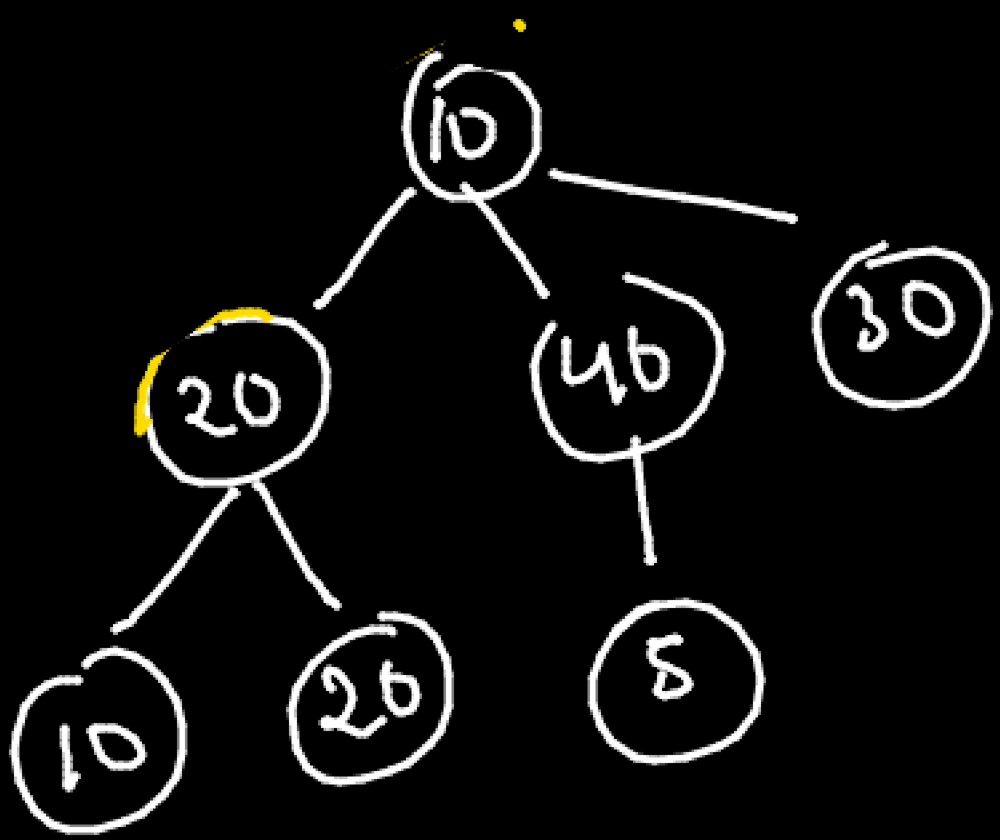
pendr



Recursion code

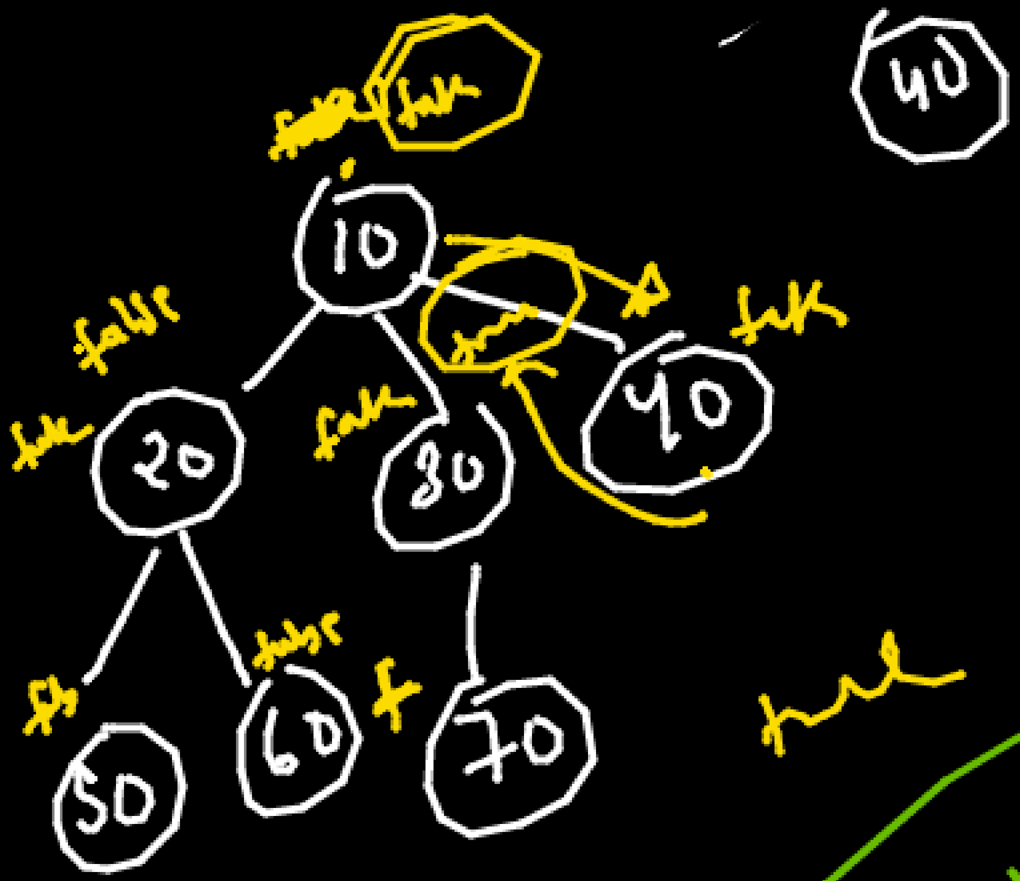
↓ queue wali easy hai

front



```
TreeNode<int>* Max( TreeNode <int>* root> {
        if ( root = = NULL) return NULL
    TreeNode <int>* curr = root;
   for (int i=0; i< root.children.size(); i++){
TreeNode<int>* tmp= Max (root ->children[i]);
        if (curr->data < temp -> data)
          {    curr->data = temp->data
                   curr = temp;
    }      }
   return  curr;
}
```

Contains X → Queue Sol

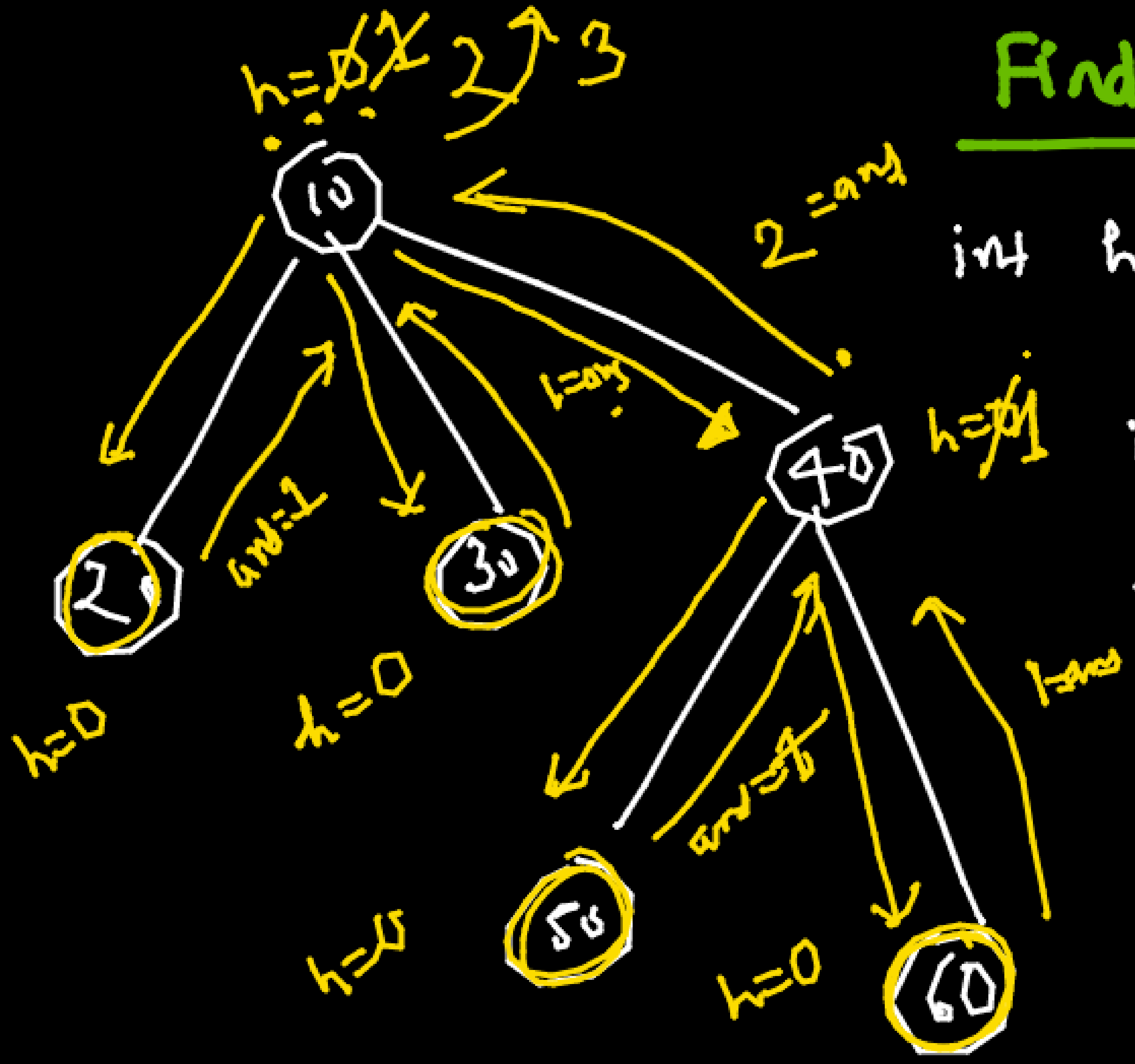


Recursion

Sol

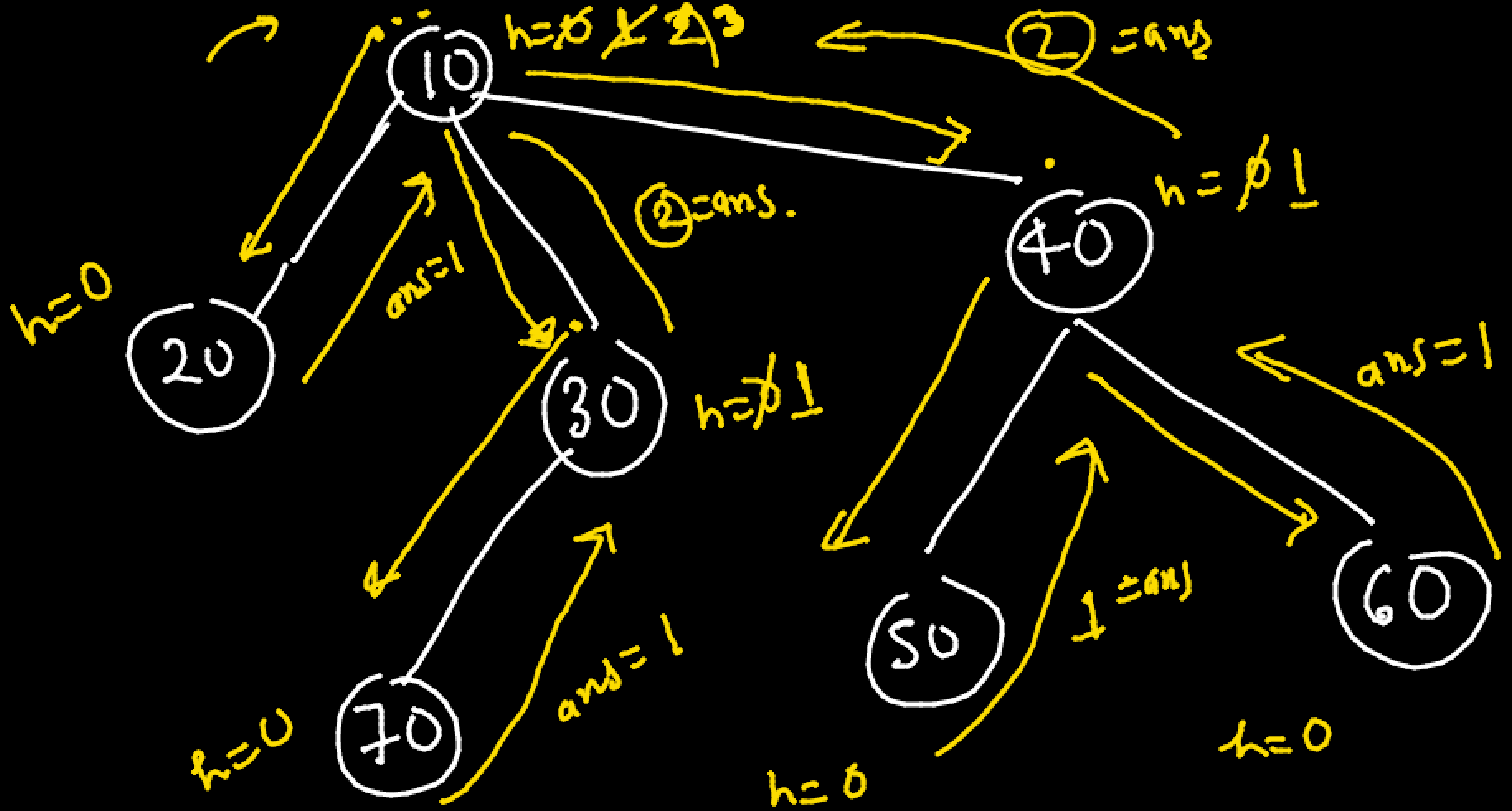
```
bool contain(TreeNode<int>* root, int x){
    bool ans = false;
    if (root -> data == x)
        return true;
for (int i=0; i< root->children.size(); i++)
{    ans = ans || contain(root->children[i], x)
}

return ans;
```

Find Height

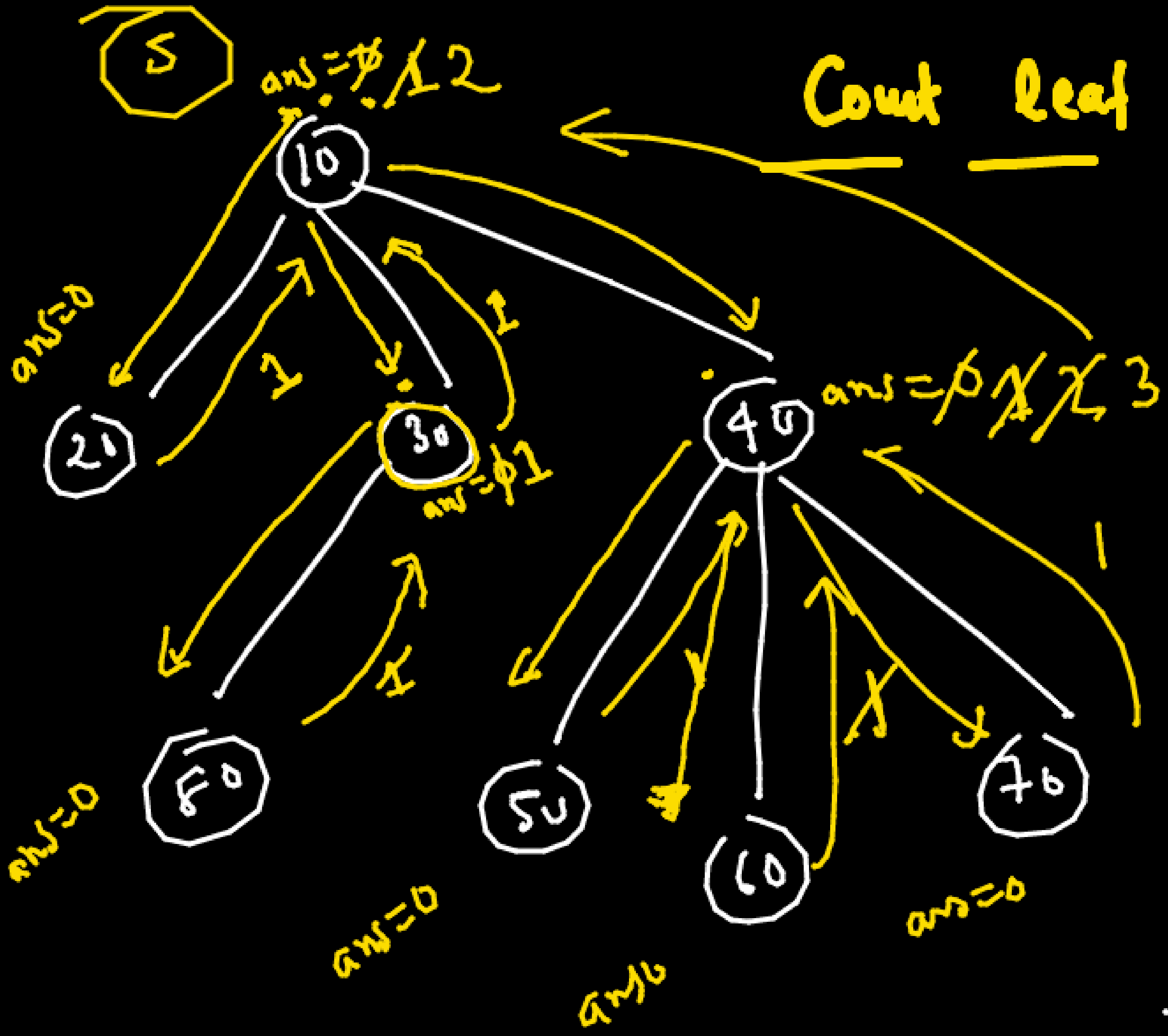


```
int height( TreeNode <int> * root) {
    if (root == NULL) return 0;
  int  h = 0;
  for (int i=0; i < root->children.size(); i++) {
       int  ans = height ( root ->children[i]);

       if (ans > h)
       {   h = ans;
       }
  }

       return 1 + h;
}
```

more Details.

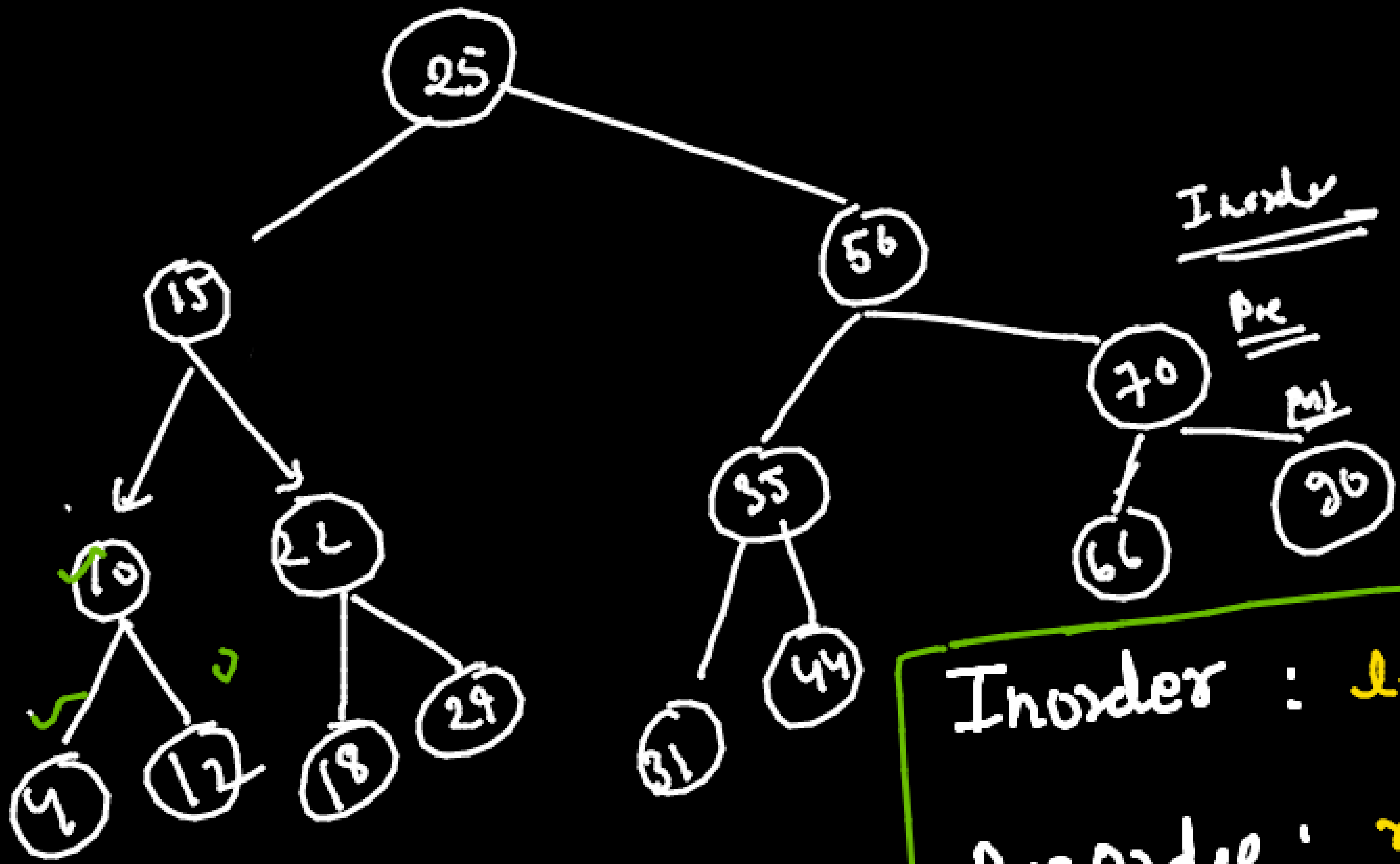

10
20
30
40
50
60
70
h=0 1 2 3
2 =ans
h=0 1
2 =ans.
h=0
ans=1
h=0 1
ans=1
h=0
ans=1
1 =ans
h=0
h=0
ans=1

Count leaf Node → Easy using Queue



```
int leaf (TreeNode <int> * root ) {
    int ans =0;
    if (root->children.size()==0)
        return 1;

    for (int i=0; i< root->children[i].size(); i++)
    { ans +=leaf ( root->children[i]);
    }

    return ans;

}
```

Traversal



66, 70, 90

Inorder 4, 10, 12, 15, 18, 22, 24^{25}, 31, 35, 44 50

Pre 25, 15, 10, 4, 12, 22, 18, 24, 56, 35, 31, 44 70, 66, 90

Post 4, 12, 10, 18, 24, 22, 15, 31, 44, 35 66, 90, 70

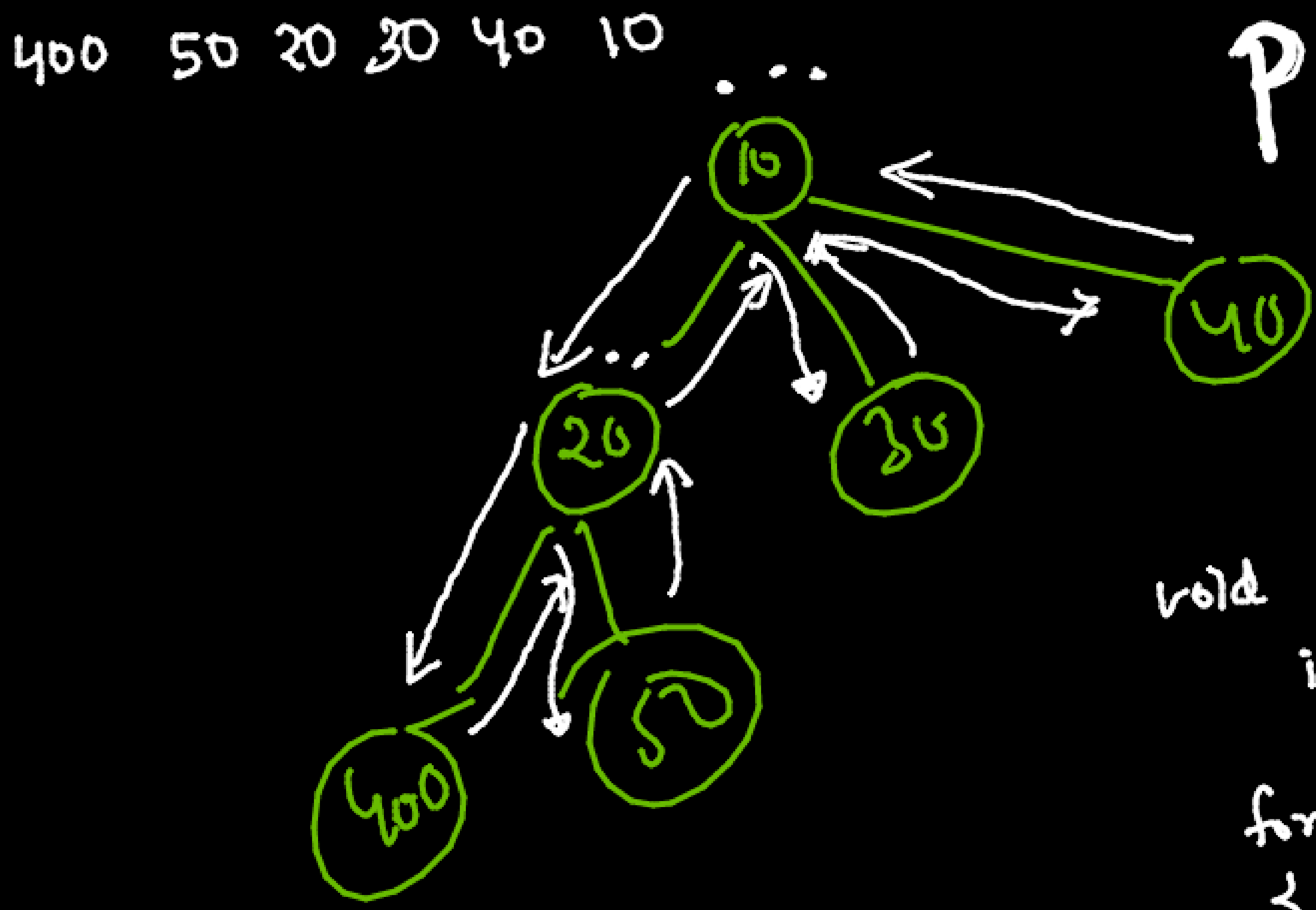
56 25

Inorder : left root right

PreOrder : root left right

PostOrder : left right root

Post Traversal

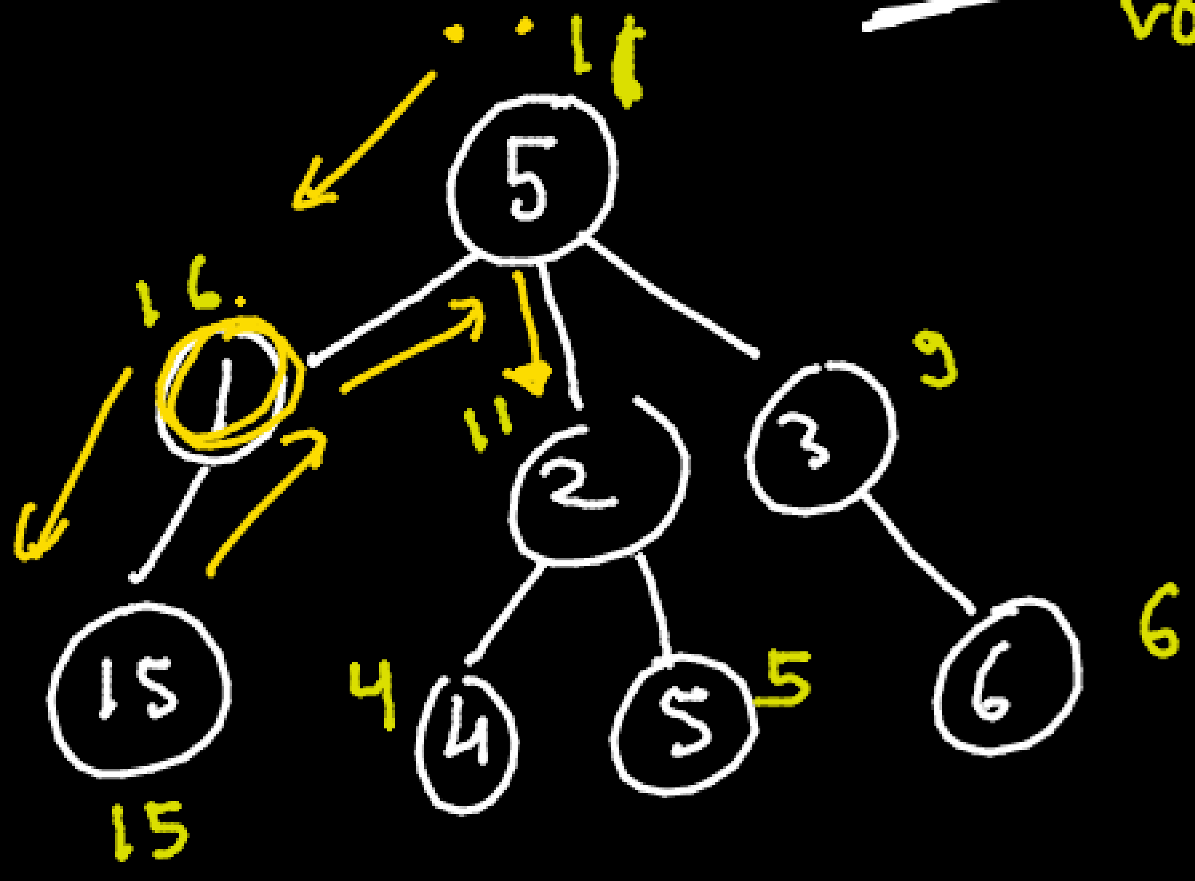


left right root

400, 50, 20, 30, 40, 10

```
void post (TreeNode<int>* root) {
    if (root == NULL) return;

  for (int i=0; i < root->children.size(); i++)
  {       post (root->children[i]);
  }
  ✓cout << root->data << " ";
}
```

Node with Max With sum

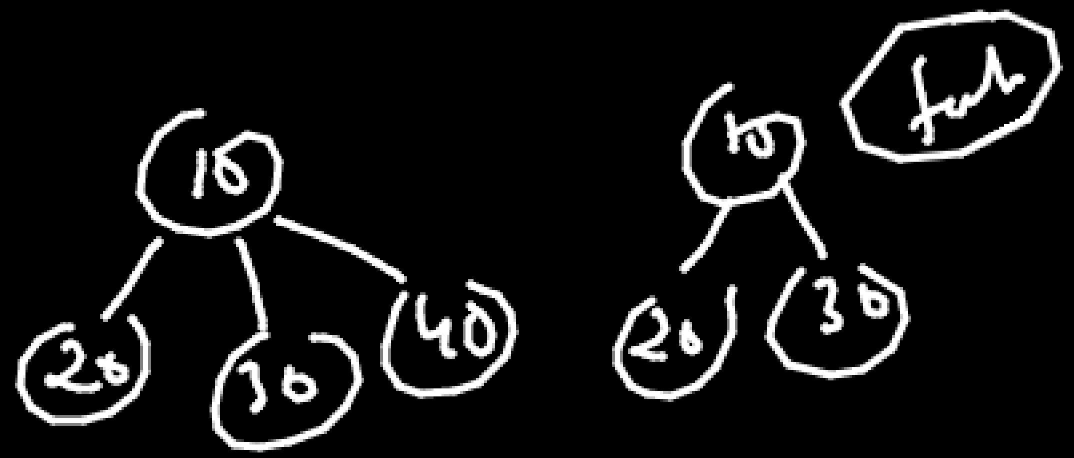


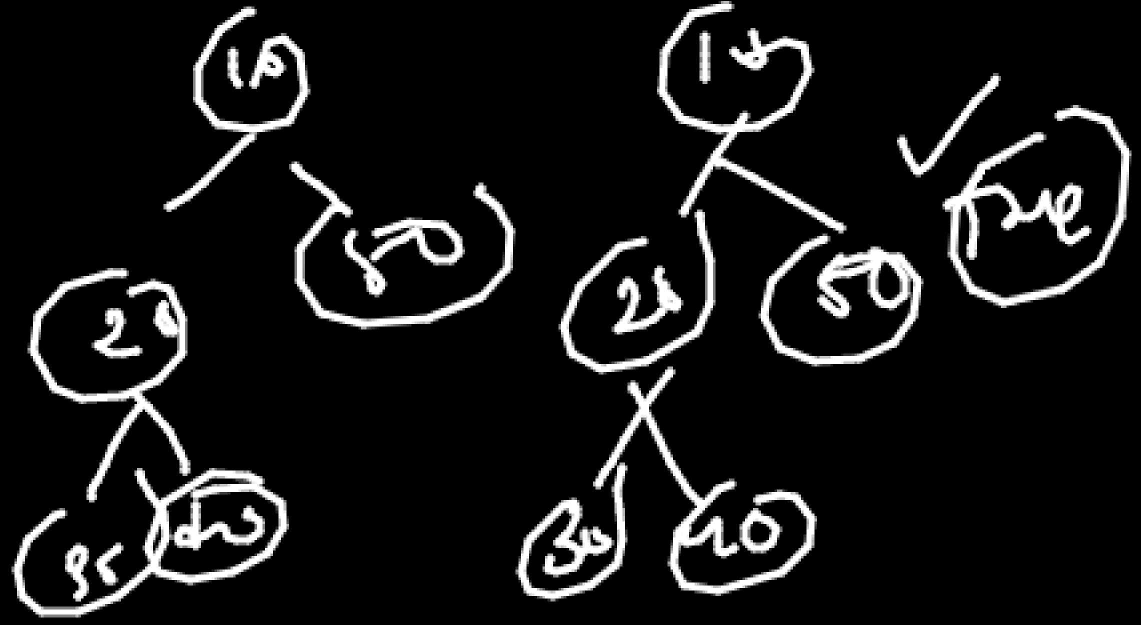
```
void fun( TreeNode <int> * root ){
    int sum = root -> data ;
  for (int i=0; i< root -> children.size(); i++)
  {     sum += root -> children [i] .data

  }
  cout << sum << endl

  for (int i=0; i<root -> children.size(); i++)
      fun( root -> children[i] );

  }
}
```

logic but Not proper code

11 9
16 6
15
11
4
5

Structually identical -





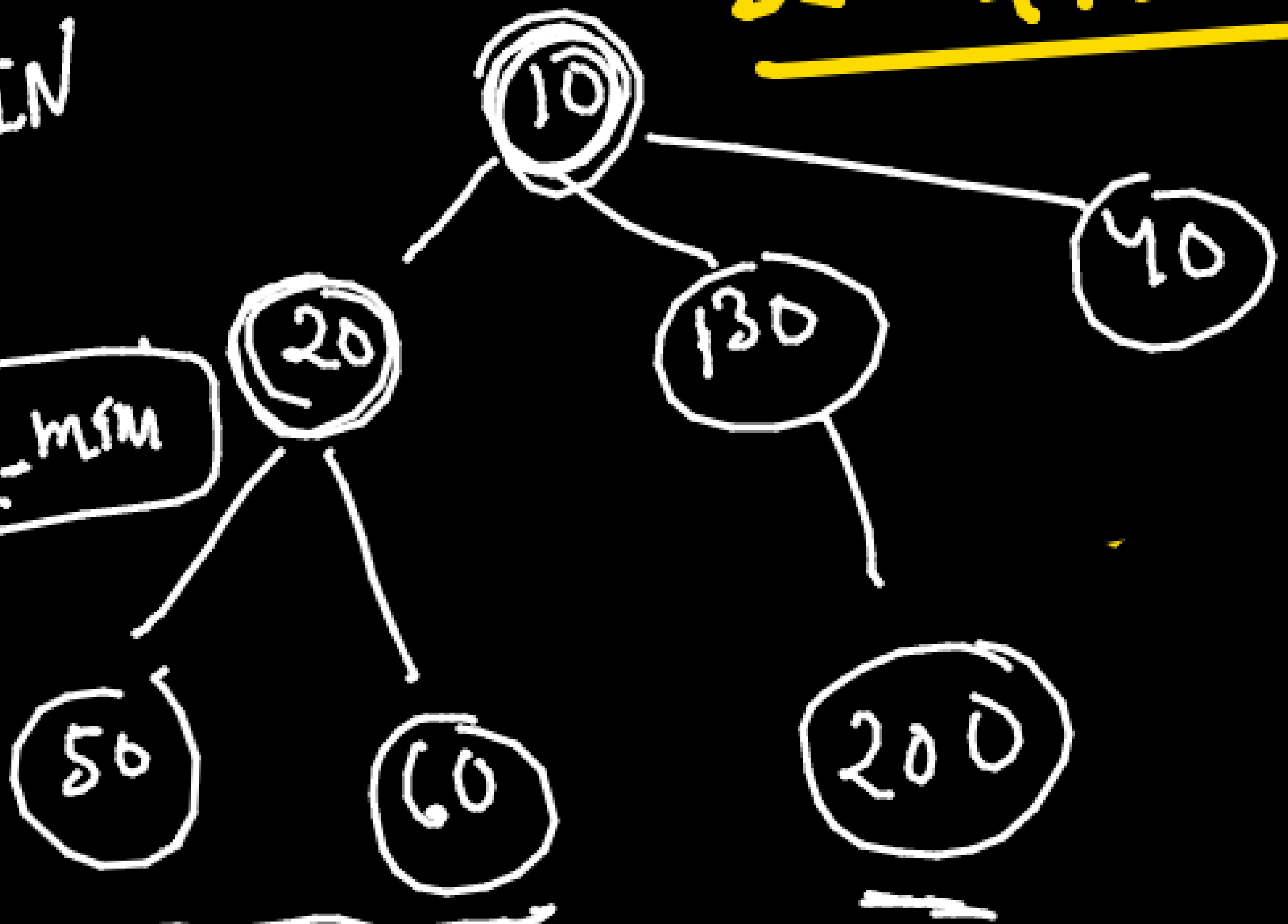
```
                                   TreeNode<int>*root2
bool areId ( root1 , root2 ) {

    if (root1 -> data != root2 -> data)
          return false;
    if( root1->children.size() != root2->children.size())
          return false

    for (int i =0; i< root->children.size(); i++){
      {
          return areId(root1->children[i], root2->children[i]
      }

  return true;
}
```

maxi

200

S_maxi

INT_MIN



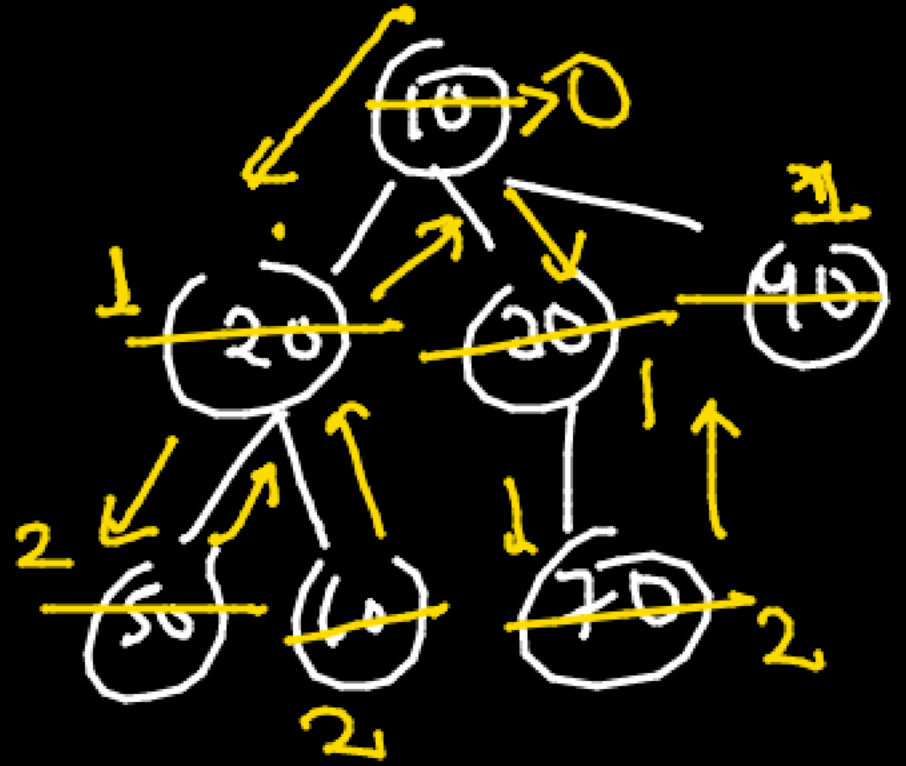
Second Max

~~20~~ 130 40

<maxi && >S_maxi

10

Replace with Depth



```
level
void replace (TreeNode <int> * root) {

        helper (root, 0);

}

void helper (TreeNode<int>* root, int level)
{
        root -> data = level
        for (int i=0, i<root->children.size();i++)
        {
                level++
                helper(root->children[i], level);
                level--
        }
}
```