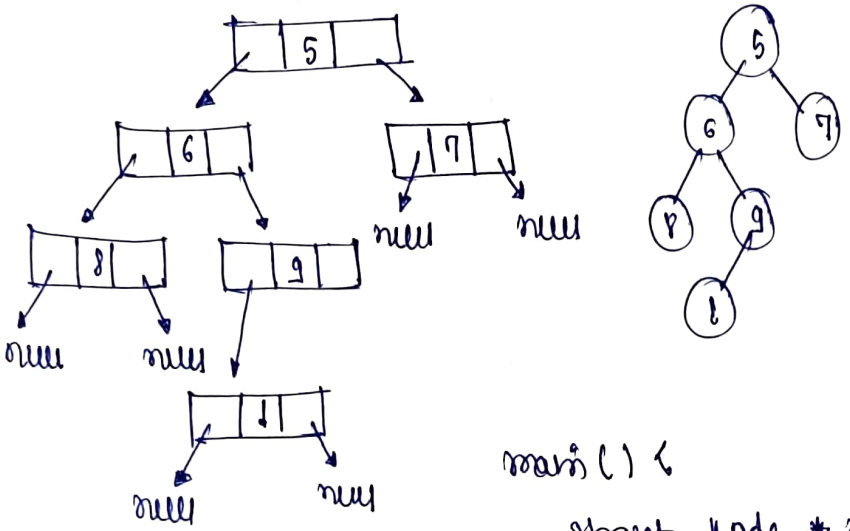
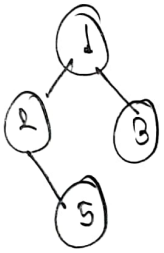
Tree

01 Binary tree representation in C++

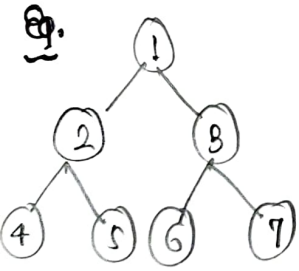



```
struct node {
    int data;
    struct node *left;
    struct node *right;
node(int val) {
    data = val;
    left = right = null;
}
main() {
    struct node *root = new node(1);
    root -> left = new node(2)
    root -> right = new node(3)
    root -> left -> right = new node(5);
}
```



02 Traversal techniques.

Eg.



→ Inorder traversal (left root right)

4 2 5 1 6 3 7

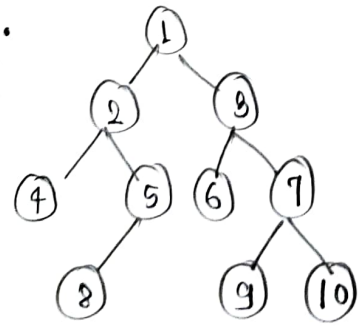
→ Pre-order traversal (root left right).

1 2 4 5 3 6 7

→ Post-order traversal (left right root)

4 5 2 6 7 3 1

Eg.



1.) Inorder : 4 2 8 5 1 6 3 9 7 10

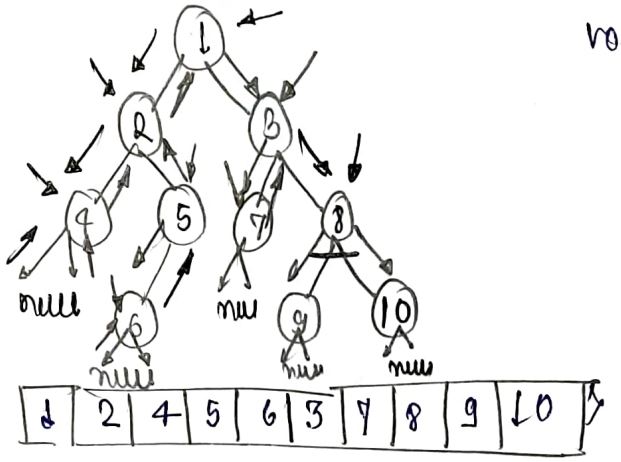
2.) Pre-order : 1 2 4 5 8 3 6 7 9 10

3.) Post-order : 4 8 5 2 6 9 10 7 3 1

Now, BFS (Breadth first search). (Level wise traversal).

1	2	3	4	5	6	7	8	9	10

03 Preorder Traversal (root left right)

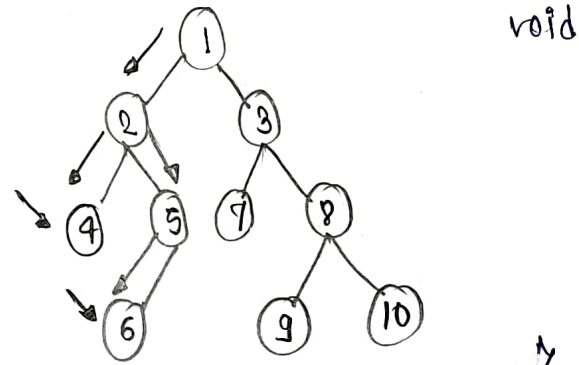


1	2	4	5	6	3	7	8	9	10

```
void preorder (node) {
    if (node == null)
    return ;
    print (node → data);
    preorder ( node → left) ;
    preorder (node → right);
```

TC = O(N)
SC = O(N)

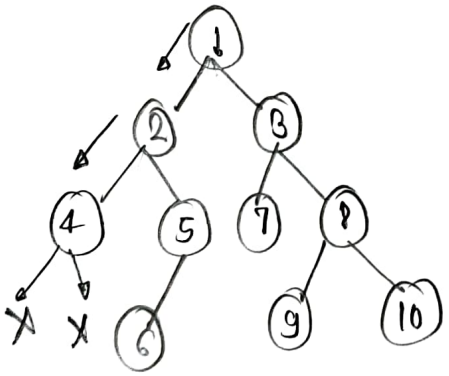
04 Inorder traversal (left root right).



4	2	6	5	1	7	3	9	8	10

```
void inorder (node) {
    if (node == null)
     return ;
    inorder (node → left).
    print ( node → data).
    inorder ( node → right).
```

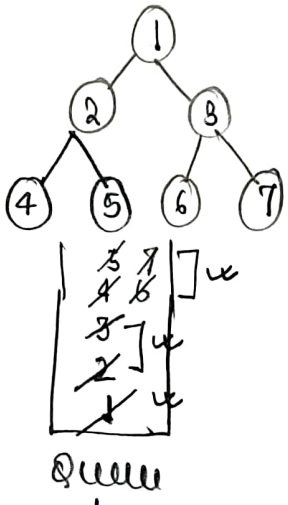
05 Postorder traversal (left right root).

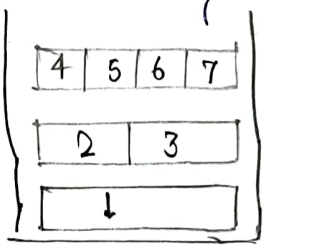


4	6	5	2	7	9	10	8	3	1

```
void postorder (node) {
    if (node == null)
     return ;
    postorder (node → left)
    post order (node → right)
     print (node → data);
```

06 Level order traversal

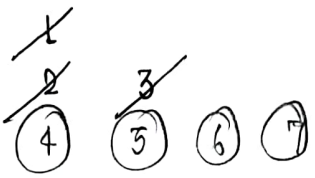




required ans.

vector of vector

```
vv<v<int>> ans;
if (root == NULL)
return ans;
queue <treeNode*> q;
q.push(root);
while(!q.empty()){
    int size = q.size();
    vector<int> level;
    for(int i=0;i<size;i++){
        treeNode* node =
            q.front();
        q.pop();
    if(node->left != NULL)
    q.push(node->left)
    if(node->right != NULL)
    q.push(node->right)
    level.push_back(node->val);
ans.push_back(level);
}
return ans;
```



Queue

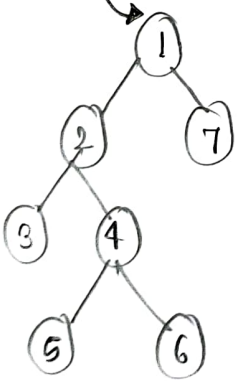
Queue becomes empty.

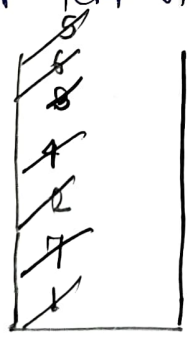
Now,

Time complexity	: O(N)
Space complexity	: O(N)

07 Iterative preorder.

(root left right)





STACK

```
vector<int> preorder;
if(root == null) return preorder;
Stack<treeNode*> st;
st.push(root);
while(!st.empty()){
    root = st.top();
    st.pop();
    preorder.push_back(root->val);
if(root->right) != NULL){
    st.push(root->right);
```

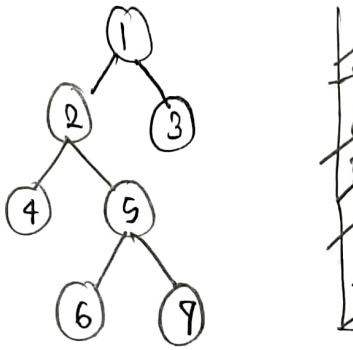
FIFO.

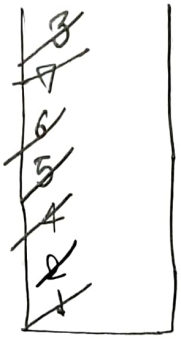
1 2 3 4 5 6 7

```
if(root->left != NULL){
    st.push(root->left);
}
}
return preorder;
```

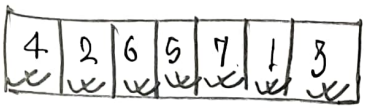
TC — O(N)
SC — O(N).

08 Iterative Inorder (left root right)

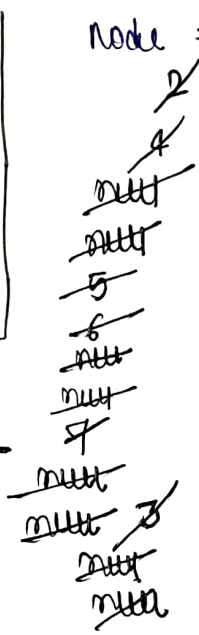




STACK LIFO

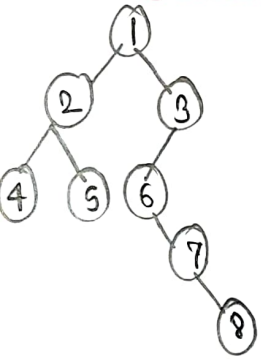


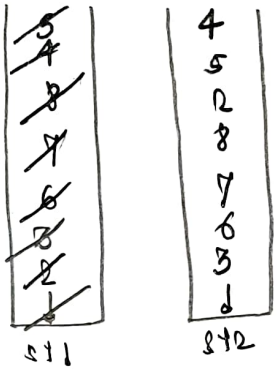
st.empty
↳ loop end.



```
stack<treenode*> st;
treeNode* node = root;
vector<int> inorder;
while (true) {
    if (node != NULL) {
        st.push(node);
        node = node->left;
    }
    else {
        if (st.empty() == true)
        break;
        node = st.top();
        st.pop();
        inorder.push(node->val);
    } node = node->right;
return inorder;
```

09 Iterative Postorder (left right root)





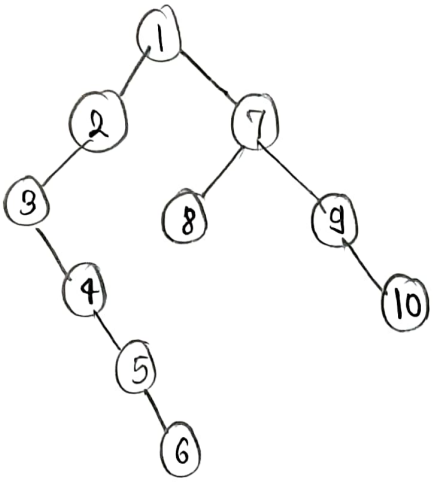
st1 st2

Now retrieve st2 element.

4 5 2 8 7 6 3 1

```
vector<int> Postorder;
if (root == NULL)
 return postorder;
stack<treenode*> st1, st2;
st1.push(root);
while (!st1.empty()) {
    root = st1.top();
    st1.pop();
    st2.push(root);
    if (root->left != NULL)
    st1.push(root->left);
}
while (!st2.empty()) {
    postorder.push_back(st2.top()->val);
    st2.pop();
}
return postorder;
```

10 Iterative postorder traversal (one stack).

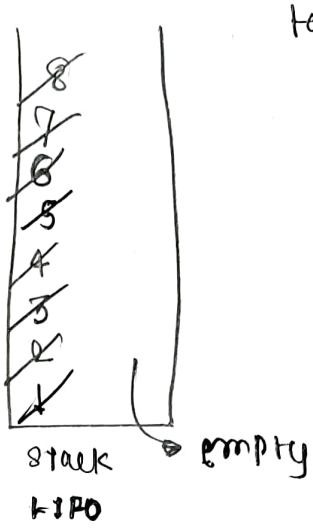


curr = ~~1~~ ~~2~~ ~~3~~ ~~null~~ ~~4~~ ~~null~~ ~~5~~ ~~null~~ ~~6~~ ~~null~~ ~~7~~ ~~8~~ null (left right root)

temp = ~~4~~ ~~5~~ ~~6~~ null
~~6~~ ~~5~~ ~~4~~ ~~3~~ null
~~2~~ ~~7~~ null
~~8~~ null
~~7~~ 1

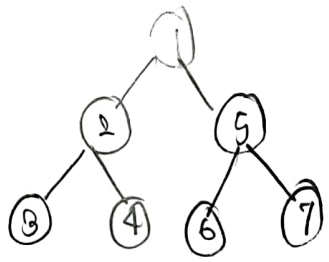
TC = O(2N)
SC = O(N)

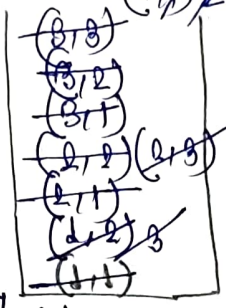
6	5	4	3	2	8	7	1




```
while (curr != NULL || !st.isEmpty()) {
    if (curr != NULL) {
        st.push(curr)
        curr = curr->left;
    }
    else {
        temp = st.top() -> right;
        if (temp == NULL) {
            temp = st.top();
            st.pop();
            post.add(temp);
            while (!st.empty() && temp == st.top() -> right) {
                temp = st.top(), st.pop();
                post.add(temp->val);
            }
        }
        else {
            curr = temp;
        }
    }
}
```

15 Postorder | Inorder | Postorder traversal





Preorder : 1 2 3 4 5 6 7 stack

Inorder : 3 2 4 1 6 5 7 (node, num)

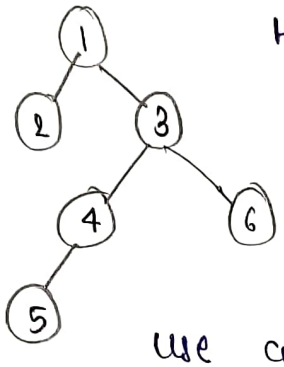
Postorder : 3 4 2 6 7 5 1

if (num = 1)
preorder, ++, left
if (num = 2)
inorder, ++, right
if (num = = 3)
postorder,

TC = O(8N)
SC = O(8N)
+ O(N)
≈ O(4N).

```
stack <pair <treeNode * int >> st;
st.push ({root, 1});
vector <int> pre, in, post;
if (root == NULL) return;
while (!st.empty ()) {
    auto it = st.top ();
    st.pop ();
    if (it.second == 1) {
        pre.push_back (it.first -> val);
        it.second++;
        st.push (it);
        if (it.first -> left != NULL) {
            st.push ({it.first -> left, 1});
    } }
    else if (it.second == 2) {
        in.push_back (it.first -> val);
        it.second++;
        st.push (it);
        if (it.first -> right != NULL) {
            st.push ({it.first -> right, 1});
    } }
    else {
        post.push_back (it.first -> val);
} }
return all the three required vectors;
```

11 Maximum depth of Binary tree



Height = 4

Recursive / Level order

O(height) — O(N)

```
if (root == NULL)
return 0;
int lh = maxdepth (root->left);
int rh = maxdepth (root->right);
return 1 + max (lh, rh);
```

main function.

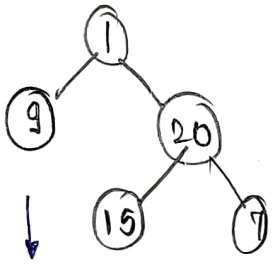
use concept

$1 + \max(l, r)$

12 Check for Balanced Binary tree

Balanced BT — For every node

$|height(left) - height(right)| <= 1$

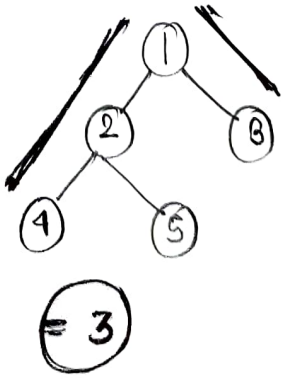


Balanced BT.

```
bool isBalanced (treeNode * root) {
    return dfsHeight (root) != -1;
}
int dfsHeight (treeNode * root) {
    if (root == NULL) return 0;
    int leftheight = dfsHeight (root->left);
    if (leftheight == -1) return -1;
    int rightheight = dfsHeight (root->right);
    if (rightheight == -1) return -1;
    if (abs (leftheight - rightheight) > 1)
    return -1;
    return max (leftheight, rightheight) + 1;
}
```

13 Diameter of Binary tree

- Longest path b/w 2 nodes
- Path does not need to pass via root



- **Naive way**

  at every node, use track of (lh + rh) and give max in out of them.

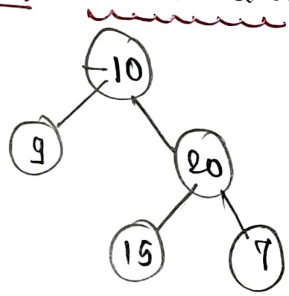
```
int dob(treenode * root){
    int diameter = 0
    height(root, diameter);
    return diameter;
}
```

```
int height(treeNode * node,
           int &diameter){
    if(!node){
        return 0;
    }
    int lh = height(node->left, diameter);
    int rh = height(node->right, diameter);
    diameter = max(diameter, lh+rh);
    return 1 + max(lh, rh);
}
```

TC = O(N)
SC = O(N).

14 maximum path sum

node A → node B

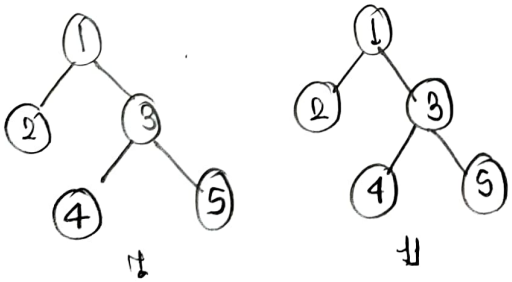


15 + 20 + 7
= (42) Ans

```
int maxPathSum(treeNode * root){
    int maxi = INT_MIN;
    maxPathDown(root, maxi);
    return maxi;
}
```

```
int maxPathDown(treeNode *
                int &maxi, root){
    if(node == NULL)
        return 0;
    int left = max(0,
        maxPathDown(node->left, maxi));
    int right = max(0,
        maxPathDown(node->right, maxi));
    maxi = max(maxi, left+right+
               node->val);
    return max(left+right) +
           node->val;
}
```

15 Same tree or Not

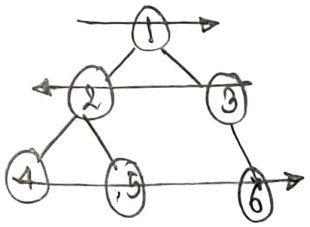


same tree

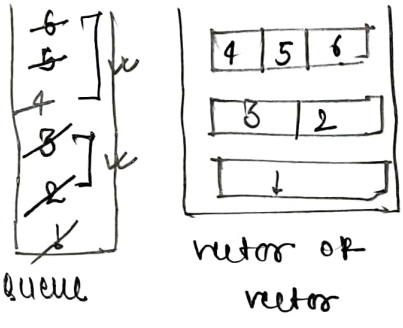
```
bool isSameTree(treeNode * p,
                treeNode * q){
    if(p == NULL || q == NULL){
        return (p == q);
    }
    return (p->val == q->val) &&
        isSameTree(p->left, q->left) &&
        isSameTree(p->right, q->right);
}
```

TC = O(N)
SC = O(N)

16 Zig-zag or spiral traversal (Adobe)



1
3 2
6 5 4



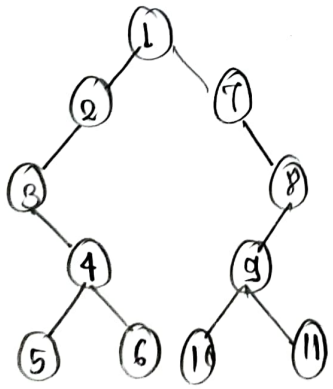
flag = ~~0~~ ~~1~~ 0 • Loop end when queue get empty

0 → L → R
1 → R → L

TC = O(N)
SC = O(N).

```
vector<vector<int>> result;
if (root == NULL) return result;
queue<TreeNode*> nodesQueue;
nodesQueue.push(root);
bool leftToRight = true;
while (!nodesQueue.empty()) {
    int size = nodesQueue.size();
    vector<int> row(size);
    for (int i=0; i<size; i++) {
        TreeNode* node = nodesQueue.front();
        nodesQueue.pop();
        // front or back wala case.
        int index = (leftToRight) ? i : (size-1-i);
        row[index] = node->val;
        if (node->left) {
            nodesQueue.push(node->left);
        }
        if (node->right) {
            nodesQueue.push(node->right);
        }
    }
    // again switch flag.
    leftToRight = !leftToRight;
    result.push_back(row);
}
return result;
```

17 Boundary traversal



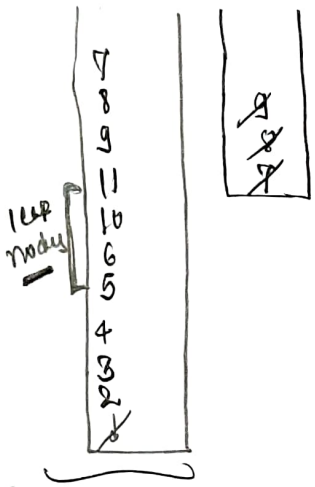
In case of ACW ↺

1 2 3 4 5 6 10 11 9 8 7

↳ How to do that ACW traversal of BT

- left boundary excluding leaf
- leaf nodes (Inorder traversal).
- right boundary reverse dirn.

↳ 3 steps process.




```
void addleft (Node * root, vector<int> & res) {
    Node * curr = root->left;
    while (curr) {
        if (!isleaf(curr))
        res.push_back (curr->data);
        if (curr->left)
         curr = curr->left;
        else
        curr = curr->right;
    }
}
```

- This is entire boundary traversal.

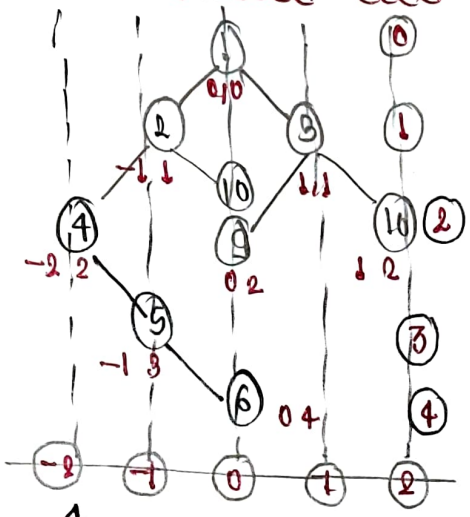
```
void addRight (Node * root, vector<int> & res) {
    Node * curr = root->right;
    vector<int> tmp;
    while (curr) {
        if (!isleaf(curr)) tmp.push_back (curr->data);
        if (curr->right) curr = curr->right;
        else curr = curr->left;
    }
    for (int i = tmp.size() - 1; i >= 0; --i) {
        res.push_back (tmp[i]);
    }
}
```

```
void addleaves (Node * root, vector<int> & res) {
    if (isleaf (root)) {
        res.push_back (root->data);
        return;
    }
    if (root->left) addleaves (root->left, res);
    if (root->right) addleaves (root->right, res);
}
```

```
vector<int> printboundary (Node * root) {
    vector<int> res;
    if (!root) return res;
    if (!isleaf (root)) res.push_back (root->data);
    addleft (root, res);
    addleaves (root, res);
    addright (root, res);
    return res;
}
```

TC = O(N)
SC = O(N).

18 Vertical order traversal.



4
2 5 → 9 10
1 10 9 6
3
10

(5, -1, 3)

~~(1,0,0)~~ ~~(2,-1,1)~~ ~~(3,1,1)~~ ~~(4,-2,2)~~ ~~(10,0,2)~~ (9,0,2) (10,1,2)

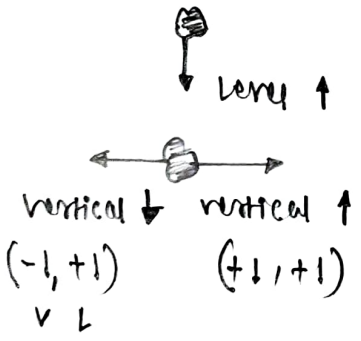
Queue (node, vertical, level)

map < int, map < int, multiset < int > > >

vertical	level	
-2	2	{4}
-1	1	{2}
0	0	{1}
1	1	{3}
0	2	{10}
⋮		

can have same value. ↓ multinode.

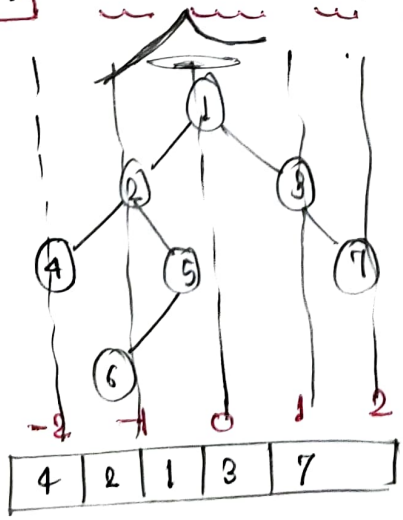
- Take queue and map accordingly



node = 1 (0,0)
node = 2 (-1,1)
node = 3 (1,1)
node = 4 (-2,2)
node = 10 (0,2)
⋮

- do this accordingly.

19 Top view of Binary Tree



- Here, we follow level order traversal
- Here we use vertical order traversal
- Har ek line (vertical) se pehla node ye to wahi top view hai

~~(6,-1)~~
~~(7,2)~~
~~(5,0)~~
~~(4,-2)~~
~~(8,1)~~
~~(2,1)~~
~~(1,0)~~
Queue (root, line)

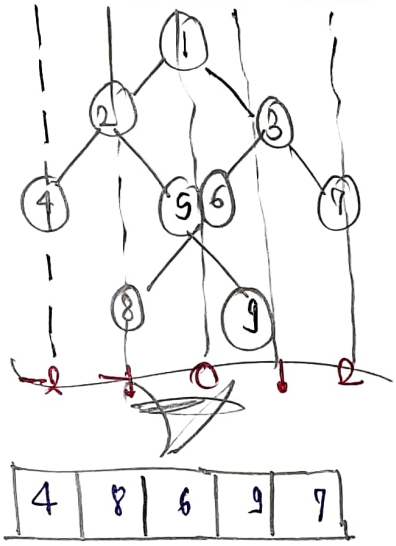
2 → 7
-2 → 4
1 → 3
-1 → 2
0 → 1
map (line, node)

node
~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~7~~
~~8~~ ~~6~~ ~~9~~ ~~10~~ ~~11~~ ~~12~~
5
~~-1~~

- Take Queue and traverse make note of Q (node, line) and also map when insert (jab line me kuch hoga nhi toh hi insert karenge nhi toh nhi karenge).

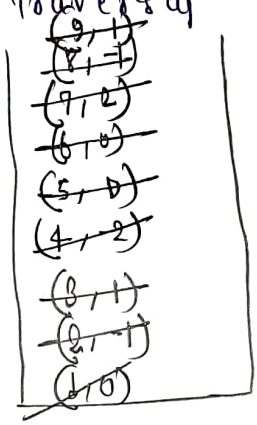
use bad map ko line ke according sort karke uska second nikal lenge vector me store kar lenge.

20 Bottom view of Binary Tree

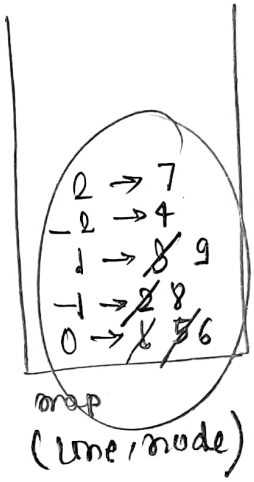


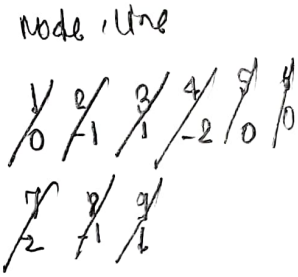
4	8	6	9	7

- Yaha hm log use karenge vertical order traversal



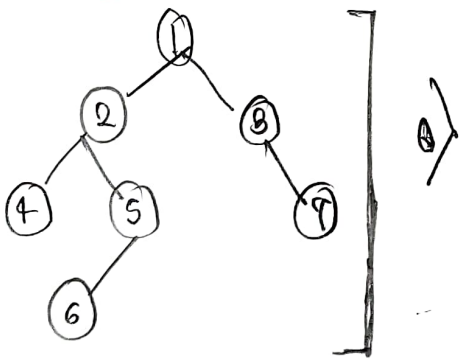
Queue (root, line)





Yaha last tak update or traverse krte jayenge map ko arrange kar denge.

21 Right / left side view



RSV

1	3	7	6

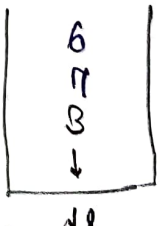
- Recursive
  - TC = O(N)
  - SC → O(H)

Reverse preorder traversal

```
R (node, level) {
    if (node == null) return;   [root right left]
    if (level == ds.size) ds.add(node)
    R(node → right, level+1);
    R(node → left, level+1);
}
```

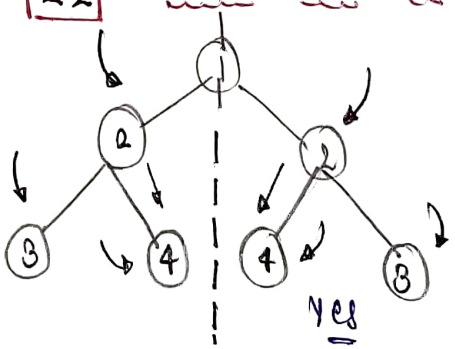
TC = O(N)
SC = O(H)

- Iterative
  - level order
    - TC = O(N)
    - SC =



Now isko kaise kr lo

22 check tree is symmetrical or not



It forms a mirror or itself around the centre.

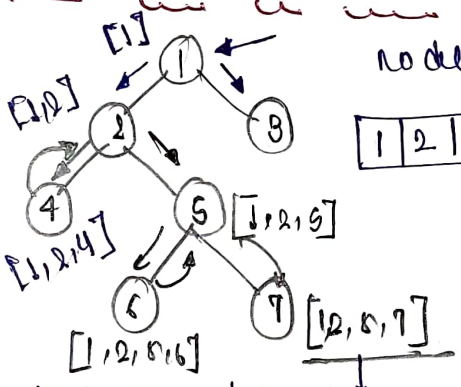
left ⇌ right

root → left | root → right

root left right | root right left

- Simply check this at any case it fails loop and return false else return true.

23 root to node path



node = 7

1	2	5	7

use recursion.

left → 1st

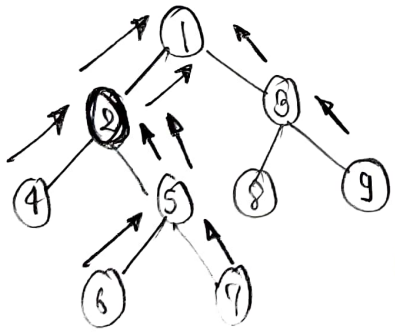
right → 2nd

TC = O(N)
SC = O(H)

- use inorder traversal
- root se start karna hai more kete jana hai node tak pahuchte pahuchte ans mil jayega.

(true aaya return ho jayenge peche return kar denge ans).

24 lowest common ancestor (at deepest level).

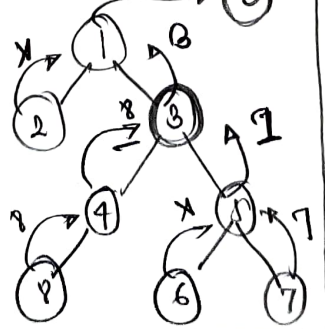


Now, use traversal techniques,

- traverse kro ek mila or dusra mila phir

LCA (4,7) = 2
LCA (5,9) = 1
LCA (2,6) = 2

eg.



LCA (7,8) - 3

node = 4 node = 7

[1, 2, 4] [1, 2, 5, 7]

path

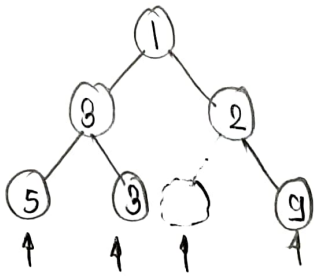
So return 2

↳ Brute force way

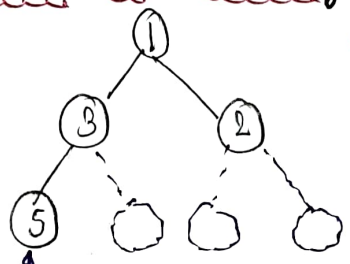
TC = O(N) + O(N) + O(N) + O(N)

- ek r taraf se out come aaya usko le lenge

29 Maximum width of Binary tree



Brute - width = 4



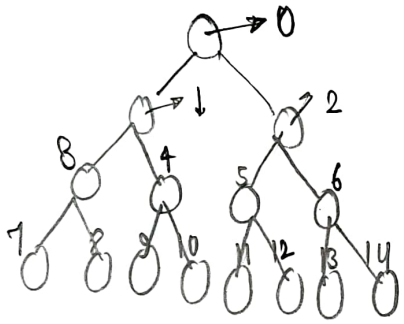
right me nodes

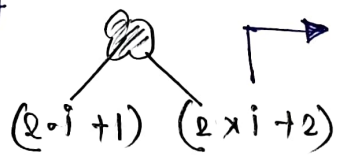
So, width = 2

- width matter karega ek level pe first node or last node ke beech me kitna node absent ho jayega ya hai

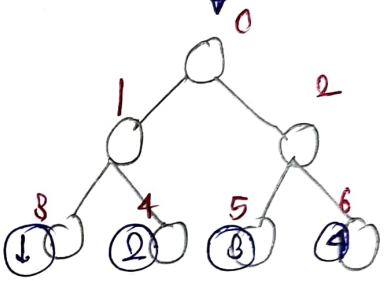
(use level order traversal).

- sab pe take indexing ki do phir sahi ans aa jayega





case of overflow occurs in that case

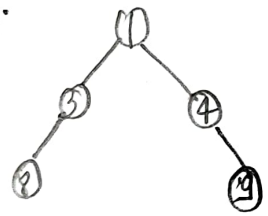


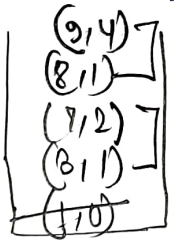
ans = Last node index − First node index + 1

$i = (i - \text{min})$

$2i+1$ $2i+2$

Eg.



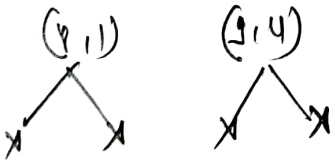


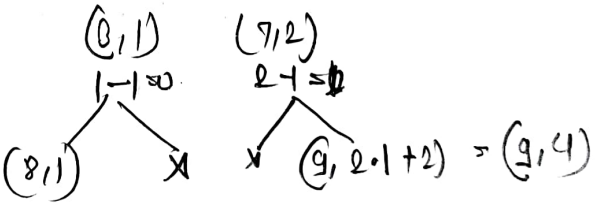
queue

width = 4

TC = O(N)

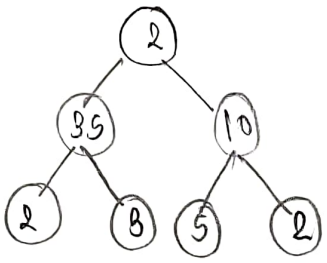
SC = O(N)

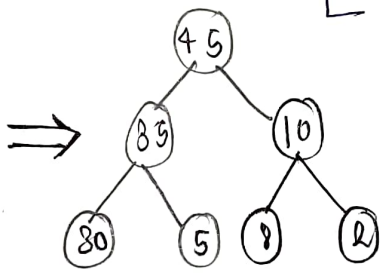




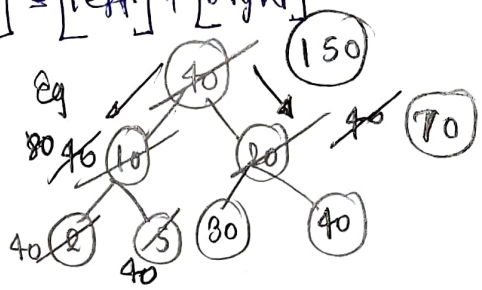
26 Children Sum Property

[node] = [left] + [right]





80+5 8+2



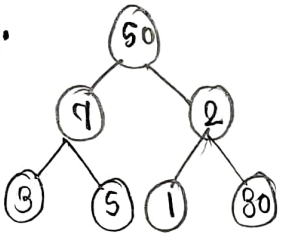
Eg

10 + 20 = 30 < 40

2 + 5 = 7 < 10

30 + 40 = 70 > 40

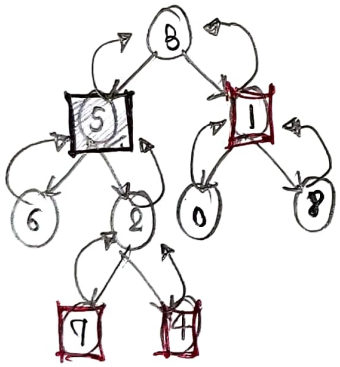
Eg.



⇒ Neeche jane pe kam milega toh node ko usi se assing kr denge opar aane pe add kr denge and jada wha upar aane pe sum toh usko same rhene denge.

increase by +1 as many times

27 Node at a distance K



k = 2

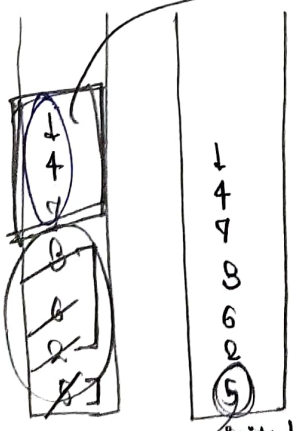
target = 5

return {7, 4, 1}

↳ do BFS traversal

Queue

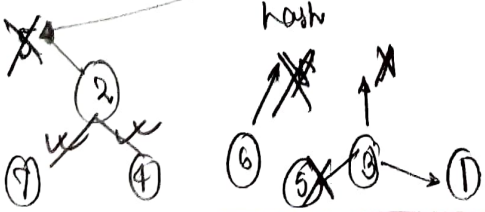
- Here we mark out the parents pointer. Now find node at distance (k).
- agar seedha node dega to kol v traversal kar ke aage bdhenge phir address nhi jayega.



Queue

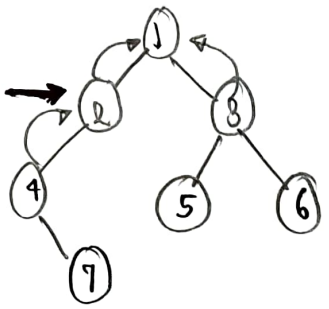
Visited hash

are yaha pe Parent Pointer ka dhyan rkhna hai



dis = 0 1 2

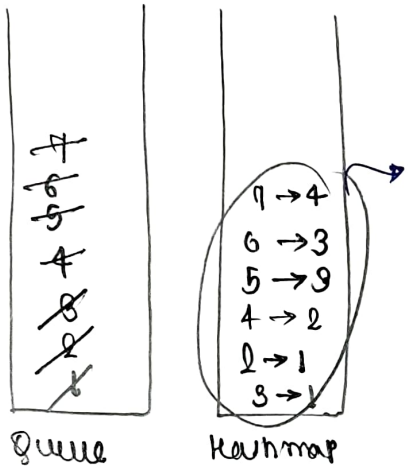
28 Min time taken to burn a BT from a node / leaf node

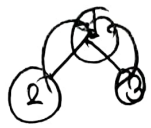


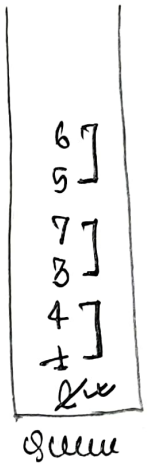
Node = 2 ↳ use BFS traversal

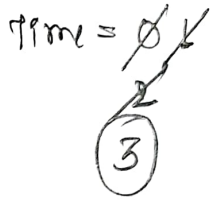
- Yaha us node se radially outward jana hai

1.) assign parent pointers.





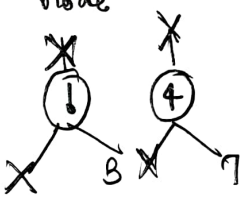


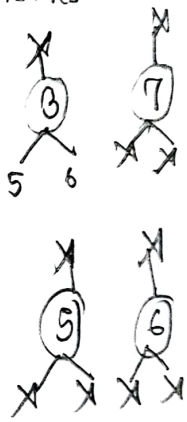


→ Why not DFS ?

Here we use level wise the things, level wise movement was there so we can't use DFS.

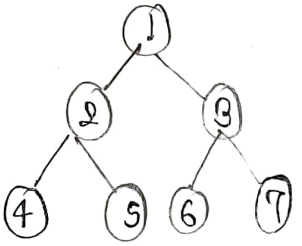
Now pick both node





- Queue will get empty then time will be the answer.

29 Count complete tree Nodes

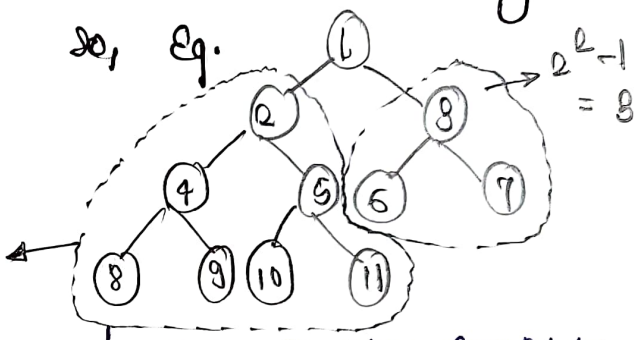


- We any other traversal and make count or that so we get
  - TC = O(N)
  - SC = O(H)
  - ↳ Brute force way
- Height of BT in worst case will be O(Log N).

Now, we property of complete tree so, we found height (level) so we use $2^3 - 1 = 7$

$\Rightarrow$ not use directly but we use smartly.

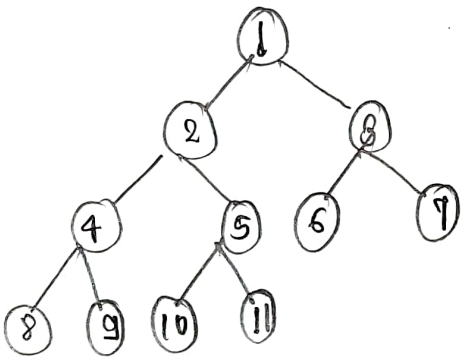
So, Eg.



↳ this is also complete tree.

Now total = 1 + 7 + 3 = 11 nodes

No. of nodes = $2^3 - 1$



- check for every subtree, any subtree is OPR full complete tree then we add again and get the answer.

at 1, we count
lh = 4
rh = 8
✗ don't use Formula.

Now traverse right and left and add it and give the ans.

at 2, lh = 3, rh = 3 | nodes = $2^3 - 1$ = 7

directly return

left will be done

at 3, lh = 2, rh = 2 | Nodes = $2^2 - 1$ = 3

directly return = 3.

Now, right will be done

1 + 7 + 3 = 11 Ans

TC = $O((\log n)^2)$ so,

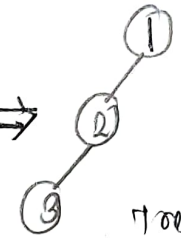
SC =

30 Requirements needed to construct unique BT

a) Preorder and Postorder

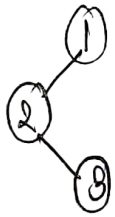
Pre order — 1 2 3 (root left right) ⇒

Post order — 3 2 1 (left right root).



Pre = 1 2 3
Post = 3 2 1

Tree 1



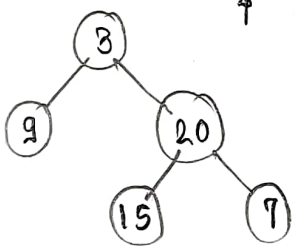
Pre = 1 2 3
Post = 3 2 1

↳ It does not generate unique BT.

b) Inorder and Preorder (Yes we create unique BT).

Inorder : 9 3 15 20 7 (left root right)

Preorder : 3 9 20 15 7 (root left right).



⇒ unique BT.

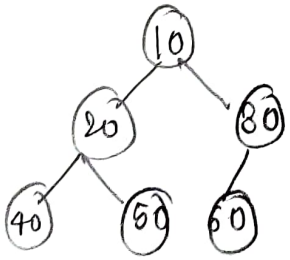
c) Inorder and Postorder (Yes we create). [unique BT].

31 construct a BT from Preorder and Inorder traversal

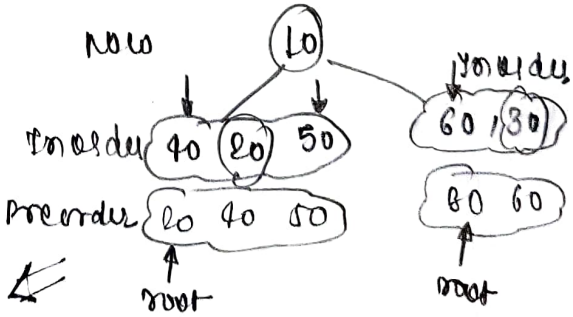
Eg. Inorder = 40 20 50 10 60 30 [left root right)

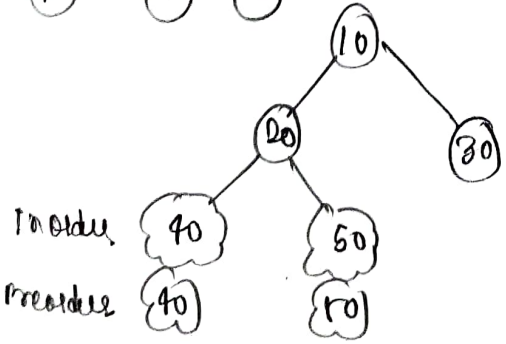
Preorder = 10 20 40 50 30 60] (root left right).

↳ always root.

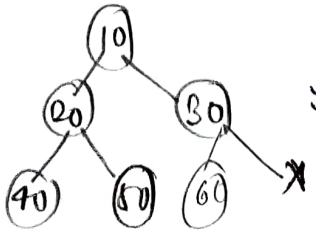


Now

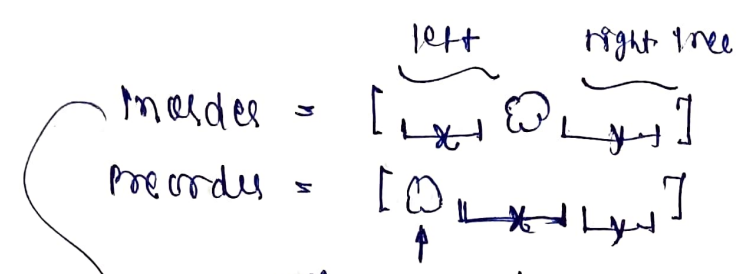




⇒



⇒ Ultimate ans.



Preorder ka first element root hai usko pakar ke inorder me dhundh lo

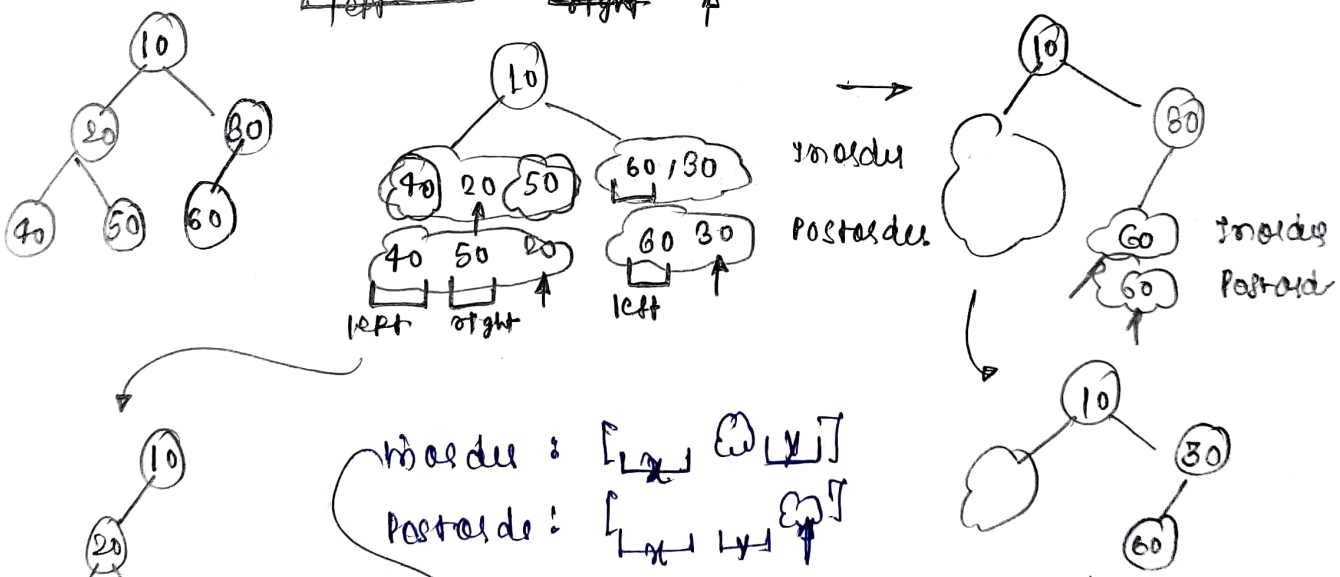
↳ ab ye naya ques ban gya x element or y element ke liye usme & same x ka pehla element root hoga and all, do this again and again.

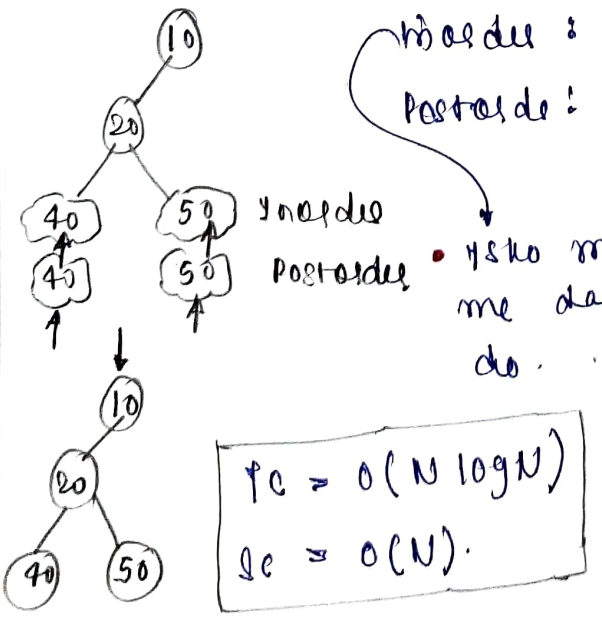
Inorder ke same value ko hash map me dal do

TC = O(N)
↳ TC = O(N log N)
SC = O(N)

32 construct a BT from postorder and inorder traversal

Inorder = [40 20 50 | 10 | 60 30] → (left root right)
Postorder = [40 50 20 | 60 30 | 10] → (left right root)



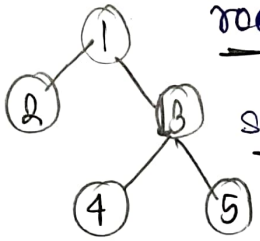


- ysko map me dal do.
- inorder ke left wala postorder ke first se x element count krenge phir naya ques phir same ke lenge
- or y element inorder se right wala hai toh postorder me x ke bad rest y element hoga phir naya ques.

TC = O(N log N)
SC = O(N).

done

33 Serialize and De-serialize BT

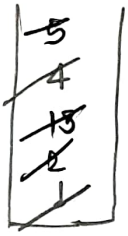


root → serialize → string → de-serialize → root

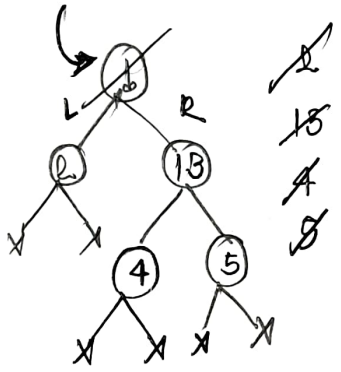
→ do it by multiple logics but we did it by level order traversal.

1, 2, 13, #, #, 4, 5, #, #, #, # ⇒ serialized string.

(2) (4) (5)



queue



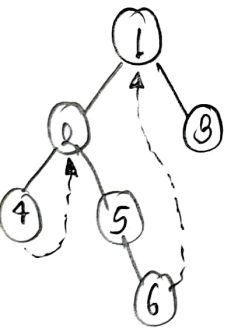
return root via serialized string.

TC = O(N)
SC = O(N).

34 Morris traversal [inorder]

→ uses threaded BT

TC = O(N)
SC = O(1).



1st left → null root (1) → right

2nd before moving left, rightmost guy on left subtree

curr = curr → left

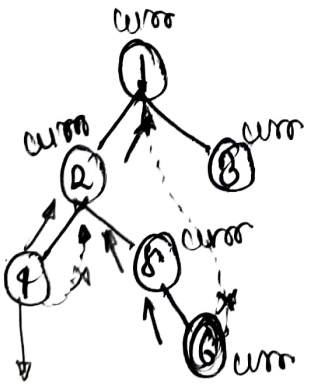
curr (if already exists).

↳ remove thread.

curr = curr → right

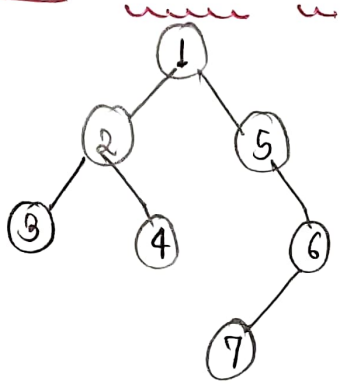
op: 4 2 5 6 1 3

eg.



4 2 5 6 1 3.

35 Flatten a BT to Linked List



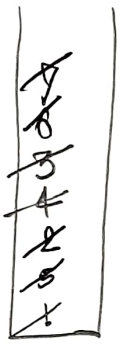
- don't create new linked list rearrange this to make a linked list.
- use traversal [Right left root] : reverse Postorder.
- do accordingly wta that code, it will be arranged.

pre: 1 2 3 4 5 6 7

```
Prev = NULL
Flatten (node) {
    if (node == 0)
      return ;
    Flatten (node->right)
    Flatten (node->left)
    node->right = prev;
    node->left = NULL;
    prev = node
}
```

TC = O(N)
SC = O(N).

2nd approach



stack LIFO

```
curr = 1 2 3 4 5 6 7
st.push (root)
while (!st.empty()) {
    curr = st.top()
    st.pop()
    if (curr->right)
    st.push (curr->right)
    if (curr->left)
    st.push (curr->left)
    if (!st.empty())
    curr->right = st.top();
    curr->left = NULL;
}
```

TC = O(N)
SC = O(N).

3rd approach

```
curr = 1 2 3 4 5 6 7 null
prev = 2 4 5 7
curr = root
while (curr != NULL) {
    if (curr->left != NULL) {
        prev = curr->left;
        while (prev->right) {
            prev = prev->right;
        }
        prev->right = curr->right;
        curr->right = curr->left;
    }
    curr = curr->right;
}
```

TC = O(N)
SC = O(1).

TREE BEST QUESTIONS

1.) INORDER TRAVERSAL

CODE

```
class Solution {
public:
    void func(TreeNode* root, vector<int> &ans){
        if(root==NULL)
        return;
        func(root->left, ans);
        ans.push_back(root->val);
        func(root->right, ans);
    }

    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> ans;
        func(root, ans);
        return ans;
    }
};
```

2.) PREORDER TRAVERSAL

CODE

```
class Solution {
public:
    void func(TreeNode* root, vector<int> &ans){
        if(root==NULL)
        return;
        ans.push_back(root->val);
        func(root->left, ans);
        func(root->right, ans);
    }

    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> ans;
        func(root, ans);
        return ans;
    }
};
```

TREE BEST QUESTIONS

3.) POSTORDER TRAVERSAL

CODE

```
class Solution {
public:
    void func(TreeNode* root, vector<int> &ans){
        if(root==NULL)
        return;
        func(root->left, ans);
        func(root->right, ans);
        ans.push_back(root->val);
    }

    vector<int> postorderTraversal(TreeNode* root) {
        vector<int> ans;
        func(root, ans);
        return ans;
    }
};
```

4.) N-ARY PREORDER TRAVERSAL

CODE

```
class Solution {
public:
    void func(Node* root, vector<int> &ans){
        if(root==NULL)
        return;
        ans.push_back(root->val);
        for(int i=0;i<root->children.size();i++){
            func(root->children[i], ans);
        }
    }

    vector<int> preorder(Node* root) {
        vector<int> ans;
        func(root, ans);
        return ans;
    }
};
```

TREE BEST QUESTIONS

5.) N-ARY POSTORDER TRAVERSAL

CODE

```
class Solution {
public:
    void func(Node* root, vector<int> &ans){
        if(root==NULL)
        return;
        for(int i=0;i<root->children.size();i++){
            func(root->children[i], ans);
        }
        ans.push_back(root->val);
    }
    vector<int> postorder(Node* root) {
        vector<int> ans;
        func(root, ans);
        return ans;
    }
};
```

TREE BEST QUESTIONS

6.) LEVEL ORDER TRAVERSAL

CODE

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> ans;

        if(root==NULL)
        return ans;

        queue<TreeNode*> q;
        q.push(root);

        while(!q.empty()){
            int size=q.size();
            vector<int> level;
            for(int i=0;i<size;i++){
                TreeNode* node=q.front();
                q.pop();
                if(node->left!=NULL)
                q.push(node->left);
                if(node->right!=NULL)
                q.push(node->right);
                level.push_back(node->val);
            }
            ans.push_back(level);
        }
        return ans;
    }
};
```

TREE BEST QUESTIONS

7.) N-ARY LEVEL ORDER TRAVERSAL

CODE 01

```
class Solution {
public:
    vector<vector<int>> levelOrder(Node* root) {
        vector<vector<int>> ans;
        if(root==NULL)
        return ans;

        queue<Node*> q;
        q.push(root);

        while(!q.empty()){
            int size=q.size();
            vector<int> level;

            for(int i=0;i<size;i++){
                Node* node=q.front();
                q.pop();
                level.push_back(node->val);
                for(auto it:node->children){
                    q.push(it);
                }
            }
            ans.push_back(level);
        }
        return ans;
    }
};
```

CODE 02

```
class Solution {
public:
    vector<vector<int>> levelOrder(Node* root) {
        vector<vector<int>> ans;
        if(root==NULL)
        return ans;

        queue<Node*> q;
        q.push(root);

        while(!q.empty()){
            int size=q.size();
            vector<int> level;
```

```
            for(int i=0;i<size;i++){
                Node* node=q.front();
                q.pop();
                for(auto it:node->children){
                    q.push(it);
                }
                level.push_back(node->val);
            }
            ans.push_back(level);
        }
        return ans;
    }
};
```

TREE BEST QUESTIONS

8.) MAXIMUM DEPTH OF BINARY TREE

CODE

```
class Solution {
public:
    int func(TreeNode* root){
        if(root==NULL)
        return 0;
        int lh=func(root->left);
        int rh=func(root->right);
        return 1+max(lh, rh);
    }

    int maxDepth(TreeNode* root) {
        return func(root);
    }
};
```

9.) MAXIMUM DEPTH OF N-ARY TREE

CODE

```
class Solution {
public:
    int func(Node* root){
        if(root==NULL)
        return 0;

        if(root->children.size()==0)
        return 1;

        vector<int> arr;
        for(int i=0;i<root->children.size();i++){
            arr.push_back(func(root->children[i]));
        }

        int depth=1+*max_element(arr.begin(), arr.end());
        return depth;
    }
    int maxDepth(Node* root) {
        return func(root);
    }
};
```

TREE BEST QUESTIONS

10.) MINIMUM DEPTH OF BINARY TREE

CODE

```
class Solution {
public:
    int func(TreeNode* root){
        if(root==NULL)
        return 0;

        if(root->left==NULL && root->right!=NULL)
        return 1+func(root->right);

        else if(root->right==NULL && root->left!=NULL)
        return 1+func(root->left);

        int lh=func(root->left);
        int rh=func(root->right);
        return 1+min(lh, rh);
    }
    int minDepth(TreeNode* root) {
        return func(root);
    }
};
```

TREE BEST QUESTIONS

11.) CHECK FOR BALANCED BINARY TREE

CODE

```
class Solution {
public:
    int height(TreeNode* root){
        if(root==NULL)
        return 0;

        int lh=height(root->left);
        if(lh==-1)
        return -1;

        int rh=height(root->right);
        if(rh==-1)
        return -1;

        if(abs(lh-rh)>1)
        return -1;

        return 1+max(lh, rh);
    }
    bool isBalanced(TreeNode* root) {
        if(height(root)==-1)
        return false;
        else
        return true;
    }
};
```

TREE BEST QUESTIONS

12.) DIAMETER OF BINARY TREE

CODE

```
class Solution {
public:
    int height(TreeNode* root, int& diameter){
        if(root==NULL)
        return 0;

        int lh=height(root->left, diameter);
        int rh=height(root->right, diameter);

        diameter=max(diameter, lh+rh);
        return 1+max(lh,rh);
    }
    int diameterOfBinaryTree(TreeNode* root) {
        int diameter=0;
        height(root, diameter);
        return diameter;
    }
};
```

13.) MAXIMUM PATH SUM

CODE

```
class Solution {
public:
    int func(TreeNode* root, int &maxi){
        if(root==NULL)
        return 0;

        int left=max(0, func(root->left, maxi));
        int right=max(0, func(root->right, maxi));

        maxi=max(maxi, (left+right+root->val));
        return root->val+max(left, right);
    }
    int maxPathSum(TreeNode* root) {
        int maxi=INT_MIN;
        func(root, maxi);
        return maxi;
    }
};
```

TREE BEST QUESTIONS

14.) SAME TREE

CODE

```
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if(p==NULL || q==NULL)
        return p==q;
        return (p->val==q->val) &&
                isSameTree(p->left, q->left) &&
                isSameTree(p->right, q->right);
    }
};
```

TREE BEST QUESTIONS

15.) ZIG ZAG TRAVERSAL OR SPIRAL TRAVERSAL

CODE

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        if(root==NULL)
        return ans;

        queue<TreeNode*> q;
        q.push(root);

        bool flag=true;

        while(!q.empty()){
            int size=q.size();
            vector<int> row(size);

            for(int i=0;i<size;i++){
                TreeNode* node=q.front();
                q.pop();

                int index=0;
                if(flag==true)
                index=i;
                else
                index=size-1-i;

                row[index]=node->val;

                if(node->left!=NULL)
                q.push(node->left);
                if(node->right!=NULL)
                q.push(node->right);
            }

            flag=!flag;
            ans.push_back(row);
        }
        return ans;
    }
};
```

TREE BEST QUESTIONS

16.) BOUNDARY TRAVERSAL

CODE

```
class Solution {
public:
    bool isLeaf(Node* root) {
        return !root->left && !root->right;
    }

    void addLeftBoundary(Node* root, vector<int>& res) {
        Node* curr = root->left;

        while (curr) {
            if (!isLeaf(curr)) {
                res.push_back(curr->data);
            }
            if (curr->left) {
                curr = curr->left;
            } else {
                curr = curr->right;
            }
        }
    }

    void addRightBoundary(Node* root, vector<int>& res) {
        Node* curr = root->right;
        vector<int> temp;

        while (curr) {
            if (!isLeaf(curr)) {
                temp.push_back(curr->data);
            }
            if (curr->right) {
                curr = curr->right;
            } else {
                curr = curr->left;
            }
        }

        for (int i = temp.size() - 1; i >= 0; --i) {
            res.push_back(temp[i]);
        }
    }

    void addLeaves(Node* root, vector<int>& res) {
        if (isLeaf(root)) {
            res.push_back(root->data);
```

```
            return;
        }
        if (root->left) {
            addLeaves(root->left, res);
        }
        if (root->right) {
            addLeaves(root->right, res);
        }
    }

    vector<int> printBoundary(Node* root) {

        vector<int> res;
        if (!root) {
            return res;
        }

        if (!isLeaf(root)) {
            res.push_back(root->data);
        }

        addLeftBoundary(root, res);
        addLeaves(root, res);
        addRightBoundary(root, res);
        return res;
    }
};
```

TREE BEST QUESTIONS

17.) VERTICAL ORDER TRAVERSAL

CODE

```
class Solution {
public:
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        map<int, map<int, multiset<int>>> nodes;

        queue<pair<TreeNode*, pair<int, int>>> todo;
        todo.push({root, {0, 0}});

        while(!todo.empty()){
            auto p=todo.front();
            todo.pop();

            TreeNode* temp=p.first;
            int x=p.second.first;
            int y=p.second.second;

            nodes[x][y].insert(temp->val);

            if(temp->left){
                todo.push({temp->left, {x-1, y+1}});
            }

            if(temp->right){
                todo.push({temp->right, {x+1, y+1}});
            }
        }

        vector<vector<int>> ans;
        for(auto p:nodes){
            vector<int> col;
            for(auto q:p.second){
                col.insert(col.end(), q.second.begin(), q.second.end());
            }
            ans.push_back(col);
        }
        return ans;
    }
};
```

TREE BEST QUESTIONS

18.) RIGHT SIDE VIEW OF BINARY TREE

CODE

```
class Solution {
public:
    void func(TreeNode* root, int level, vector<int> &ans){
        if(root==NULL)
        return;

        if(ans.size()==level){
            ans.push_back(root->val);
        }
        func(root->right, level+1, ans);
        func(root->left, level+1, ans);
    }
    vector<int> rightSideView(TreeNode* root) {
        vector<int> ans;
        func(root, 0, ans);
        return ans;
    }
};
```

19.) LEFT SIDE VIEW OF BINARY TREE

CODE

```
class Solution {
public:
    void func(TreeNode* root, int level, vector<int> &ans){
        if(root==NULL)
        return;

        if(ans.size()==level){
            ans.push_back(root->val);
        }
        func(root->left, level+1, ans);
        func(root->right, level+1, ans);
    }
    vector<int> leftSideView(TreeNode* root) {
        vector<int> ans;
        func(root, 0, ans);
        return ans;
    }
};
```

TREE BEST QUESTIONS

20.) TOP VIEW OF BINARY TREE

CODE

```
class Solution
{
    public:
    //Function to return a list of nodes visible from the top view
    //from left to right in Binary Tree.
    vector<int> topView(Node *root)
    {
        vector<int> ans;
        if(root==NULL)
        return ans;

        map<int, int> mp;

        queue<pair<Node*, int>> q;
        q.push({root, 0});

        while(!q.empty()){
            auto it=q.front();
            q.pop();
            Node* node=it.first;
            int x=it.second;

            if(mp.find(x)==mp.end()){
                mp[x]=node->data;
            }

            if(node->left!=NULL){
                q.push({node->left, x-1});
            }
            if(node->right!=NULL){
                q.push({node->right, x+1});
            }
        }
        for(auto it:mp){
            ans.push_back(it.second);
        }
        return ans;
    }

};
```

TREE BEST QUESTIONS

21.) BOTTOM VIEW OF BINARY TREE

CODE

```
class Solution {
  public:
    vector <int> bottomView(Node *root) {
        vector<int> ans;
        if(root==NULL)
        return ans;

        map<int,int> mp;

        queue<pair<Node*, int>> q;
        q.push({root, 0});

        while(!q.empty()){
            auto it=q.front();
            q.pop();
            Node* node=it.first;
            int x=it.second;

            mp[x]=node->data;

            if(node->left!=NULL)
            q.push({node->left, x-1});

            if(node->right!=NULL)
            q.push({node->right, x+1});
        }
        for(auto it:mp){
            ans.push_back(it.second);
        }
        return ans;
    }
};
```

TREE BEST QUESTIONS

22.) SYMMETRIC TREE

CODE

```
class Solution {
public:
    bool func(TreeNode* l, TreeNode* r){
        if(l==NULL && r==NULL)
        return true;

        if(l==NULL && r!=NULL || l!=NULL && r==NULL)
        return false;

        if(l->val!=r->val)
        return false;

        return func(l->left, r->right) & func(l->right, r->left);
    }
    bool isSymmetric(TreeNode* root) {
        return func(root->left, root->right);
    }
};
```

TREE BEST QUESTIONS

23.) ROOT TO LEAF ALL PATHS

CODE 01

```
class Solution {
public:
    void func(Node* root, vector<vector<int>> &ans, vector<int> &temp) {
        if (root == NULL)
            return;
        if (root->left == NULL && root->right == NULL) {
            temp.push_back(root->data);
            ans.push_back(temp);
            temp.pop_back();
            return;
        }
        temp.push_back(root->data);
        func(root->left, ans, temp);
        func(root->right, ans, temp);
        temp.pop_back();
    }

    vector<vector<int>> Paths(Node* root) {
        vector<vector<int>> ans;
        vector<int> temp;
        func(root, ans, temp);
        return ans;
    }
};
```

TREE BEST QUESTIONS

CODE 02

```
class Solution {
public:
    void func(TreeNode* root, vector<string> &ans, string t){
        if(root->left==NULL && root->right==NULL){
            ans.push_back(t);
            return;
        }
        if(root->left){
            func(root->left, ans, t+"->"+to_string(root->left->val));
        }
        if(root->right){
            func(root->right, ans, t+"->"+to_string(root->right->val));
        }
    }
    vector<string> binaryTreePaths(TreeNode* root) {
        vector<string> ans;

        if(root==NULL)
        return ans;

        func(root, ans, to_string(root->val));
        return ans;
    }
};
```

TREE BEST QUESTIONS

24.) LOWEST COMMON ANCESTOR

CODE

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(root==NULL || root==p || root==q)
        return root;

        TreeNode* left=lowestCommonAncestor(root->left, p, q);
        TreeNode* right=lowestCommonAncestor(root->right, p, q);

        if(left==NULL)
        return right;
        else if(right==NULL)
        return left;
        else
        return root;
    }
};
```

TREE BEST QUESTIONS

25.) MAX WIDTH OF BINARY TREE

CODE

```
class Solution {
public:
    int widthOfBinaryTree(TreeNode* root) {
        if(root==NULL)
        return 0;

        int ans=0;
        queue<pair<TreeNode*, long long int>> q;
        q.push({root, 0});

        while(!q.empty()){
            int size=q.size();
            int mini=q.front().second;
            int first, last;

            for(int i=0;i<size;i++){
                long long int curr=q.front().second-mini;
                TreeNode* node=q.front().first;
                q.pop();

                if(i==0)
                first=curr;
                if(i==size-1)
                last=curr;

                if(node->left)
                q.push({node->left, curr*2+1});
                if(node->right)
                q.push({node->right, curr*2+2});
            }
            ans=max(ans, last-first+1);
        }
        return ans;
    }
};
```

TREE BEST QUESTIONS

26.) SUM ROOT TO LEAF NODES

CODE

SAME AS ROOT TO LEAF NODE PATH WALA QUESTION

```
class Solution {
public:
    void func(TreeNode* root, vector<vector<int>> &ans, vector<int> &temp) {
        if (root == NULL)
            return;
        if (root->left == NULL && root->right == NULL) {
            temp.push_back(root->val);
            ans.push_back(temp);
            temp.pop_back();
            return;
        }
        temp.push_back(root->val);
        func(root->left, ans, temp);
        func(root->right, ans, temp);
        temp.pop_back();
    }

    std::string arrayToString(const std::vector<int>& numbers) {
        std::stringstream ss;
        for (const int& num : numbers) {
            ss << num;
        }
        return ss.str();
    }

    int sumNumbers(TreeNode* root) {
        vector<vector<int>> ans;
        vector<int> temp;
        func(root, ans, temp);
        int sum=0;
        for(int i=0;i<ans.size();i++){
            string s;
            s=arrayToString(ans[i]);
            sum=sum+stoi(s);
        }
        return sum;
    }
};
```

TREE BEST QUESTIONS

27.) CHECK FOR CHILDREN SUM PROPERTY

CODE

```
class Solution{
    public:
    //Function to check whether all nodes of a tree have the value
    //equal to the sum of their child nodes.
    int isSumProperty(Node *node){
        if(node==NULL)
        return 1;
        int sum=0;

        if(node->left==NULL && node->right==NULL)
        return 1;
        else{
            if(node->left!=NULL)
            sum=sum+node->left->data;
            if(node->right!=NULL)
            sum=sum+node->right->data;

            return ((node->data==sum)
                     && isSumProperty(node->left)
                     && isSumProperty(node->right));
        }
    }
};
```

TREE BEST QUESTIONS

28.) ALL NODES DISTANCE K IN BINARY TREE

CODE

```
class Solution {
public:
    void buildParentMap(TreeNode* node, TreeNode* parent,
unordered_map<TreeNode*, TreeNode*> &mp){
        if(node){
            mp[node]=parent;
            buildParentMap(node->left, node, mp);
            buildParentMap(node->right, node, mp);
        }
    }
    vector<int> distanceK(TreeNode* root, TreeNode* target, int k) {
        unordered_map<TreeNode*, TreeNode*> mp;
        buildParentMap(root, NULL, mp);

        unordered_set<TreeNode*> vis;

        queue<TreeNode*> q;
        q.push(target);
        vis.insert(target);
        int curr_dis=0;

        while(!q.empty()){
            if(curr_dis==k){
                vector<int> ans;
                while(!q.empty()){
                    ans.push_back(q.front()->val);
                    q.pop();
                }
                return ans;
            }
            int size=q.size();
            for(int i=0;i<size;i++){
                TreeNode* node=q.front();
                q.pop();

                if(node->left && vis.find(node->left)==vis.end()){
                    q.push(node->left);
                    vis.insert(node->left);
                }

                if(node->right && vis.find(node->right)==vis.end()){
                    q.push(node->right);
                    vis.insert(node->right);
                }
```

```
                if(mp[node] && vis.find(mp[node])==vis.end()){
                    q.push(mp[node]);
                    vis.insert(mp[node]);
                }
            }
            curr_dis++;
        }
        return {};
    }
};
```

TREE BEST QUESTIONS

29.) AMOUNT OF TIME FOR BINARY TREE TO BE INFECTED

CODE

```
class Solution {
public:
    TreeNode* findStartNode(TreeNode* node, int start,
unordered_map<TreeNode*, TreeNode*>& pMap) {
        if (!node) return nullptr;
        queue<TreeNode*> q;
        q.push(node);
        TreeNode* sNode = nullptr;

        while (!q.empty()) {
            TreeNode* curr = q.front();
            q.pop();

            if (curr->val == start) {
                sNode = curr;
            }

            if (curr->left) {
                pMap[curr->left] = curr;
                q.push(curr->left);
            }
            if (curr->right) {
                pMap[curr->right] = curr;
                q.push(curr->right);
            }
        }

        return sNode;
    }

    int bfs(TreeNode* sNode, unordered_map<TreeNode*, TreeNode*>& pMap) {
        unordered_set<TreeNode*> visited;
        queue<TreeNode*> q;
        q.push(sNode);
        visited.insert(sNode);
        int time = 0;

        while (!q.empty()) {
            int size = q.size();
            for (int i = 0; i < size; ++i) {
                TreeNode* node = q.front();
                q.pop();

                if (node->left && visited.find(node->left) == visited.end()) {
```

TREE BEST QUESTIONS

```
                    visited.insert(node->left);
                    q.push(node->left);
                }
                if (node->right && visited.find(node->right) == visited.end())
{
                    visited.insert(node->right);
                    q.push(node->right);
                }
                if (pMap[node] && visited.find(pMap[node]) == visited.end()) {
                    visited.insert(pMap[node]);
                    q.push(pMap[node]);
                }
            }
            if (!q.empty()) {
                ++time;
            }
        }

        return time;
    }
    int amountOfTime(TreeNode* root, int start) {
        unordered_map<TreeNode*, TreeNode*> parentMap;
        TreeNode* startNode = findStartNode(root, start, parentMap);
        return bfs(startNode, parentMap);
    }
};
```

TREE BEST QUESTIONS

30.) COUNT COMPLETE TREE NODES

CODE 01

```
class Solution {
public:
    void func(TreeNode* root, int &ans){
        if(root==NULL)
        return;
        ans=ans+1;
        func(root->left, ans);
        func(root->right, ans);
    }
    int countNodes(TreeNode* root) {
        int ans=0;
        func(root, ans);
        return ans;
    }
};
```

CODE 02

```
class Solution {
public:
    int countNodes(TreeNode* root) {
        if(root==NULL)
        return 0;
        int lh=0;
        int rh=0;
        TreeNode* l=root;
        TreeNode* r=root;

        while(l!=NULL){
            lh++;
            l=l->left;
        }
        while(r!=NULL){
            rh++;
            r=r->right;
        }

        if(lh==rh)
        return (1<<lh)-1;

        int left=countNodes(root->left);
        int right=countNodes(root->right);
        return 1+left+right;
    }
};
```

TREE BEST QUESTIONS

31.) MORRIS TRAVERSAL (INORDER)

CODE

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> ans;
        TreeNode* curr=root;

        while(curr!=NULL){
            if(curr->left==NULL){
                ans.push_back(curr->val);
                curr=curr->right;
            }
            else{
                TreeNode* prev=curr->left;
                while(prev->right!=NULL && prev->right!=curr){
                    prev=prev->right;
                }
                if(prev->right==NULL){
                    prev->right=curr;
                    curr=curr->left;
                }
                else{
                    prev->right=NULL;
                    ans.push_back(curr->val);
                    curr=curr->right;
                }
            }
        }
        return ans;
    }
};
```

TREE BEST QUESTIONS

32.) COVERT A TREE INTO ITS MIRROR

CODE

```
class Solution {
  public:
    // Function to convert a binary tree into its mirror tree.
    void mirror(Node* node) {
        if(node==NULL)
        return;

        Node* temp;
        temp=node->left;
        node->left=node->right;
        node->right=temp;

        mirror(node->left);
        mirror(node->right);
    }
};
```

TREE BEST QUESTIONS

33.) CONSTRUCT BINARY TREE FROM PREORDER AND INORDER TRAVERSAL

```
class Solution {
public:
    TreeNode* buildTreeHelper(vector<int>& preorder, int preStart, int preEnd,
                              vector<int>& inorder, int inStart, int inEnd,
                              unordered_map<int, int>& inorderMap) {
        if (preStart > preEnd || inStart > inEnd) {
            return nullptr;
        }

        int rootVal = preorder[preStart];
        TreeNode* root = new TreeNode(rootVal);

        int rootIdxInorder = inorderMap[rootVal];
        int leftSubtreeSize = rootIdxInorder - inStart;

        root->left = buildTreeHelper(preorder, preStart + 1,
                                     preStart + leftSubtreeSize, inorder,
                                     inStart, rootIdxInorder - 1, inorderMap);
        root->right = buildTreeHelper(preorder,
                                      preStart + leftSubtreeSize + 1,
                                      preEnd, inorder, rootIdxInorder + 1,
                                      inEnd, inorderMap);

        return root;
    }

    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        unordered_map<int, int> inorderMap;
        for (int i = 0; i < inorder.size(); ++i) {
            inorderMap[inorder[i]] = i;
        }
        return buildTreeHelper(preorder, 0, preorder.size() - 1, inorder, 0,
                               inorder.size() - 1, inorderMap);
    }
};
```

34.) CONSTRUCT BINARY TREE FROM INORDER AND POSTORDER TRAVERSAL

```
class Solution {
public:
    TreeNode* buildTreeHelper(vector<int>& inorder, int inStart, int inEnd,
                              vector<int>& postorder, int postStart,
                              int postEnd,
                              unordered_map<int, int>& inorderMap) {
        if (inStart > inEnd || postStart > postEnd) {
            return nullptr;
        }

        int rootVal = postorder[postEnd];
        TreeNode* root = new TreeNode(rootVal);

        int rootIdxInorder = inorderMap[rootVal];
        int leftSubtreeSize = rootIdxInorder - inStart;

        root->left = buildTreeHelper(inorder, inStart, rootIdxInorder - 1,
                                     postorder, postStart,
                                     postStart + leftSubtreeSize - 1,
                                     inorderMap);
        root->right = buildTreeHelper(inorder, rootIdxInorder + 1, inEnd,
                                      postorder, postStart + leftSubtreeSize,
                                      postEnd - 1, inorderMap);

        return root;
    }

    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        unordered_map<int, int> inorderMap;
        for (int i = 0; i < inorder.size(); ++i) {
            inorderMap[inorder[i]] = i;
        }
        return buildTreeHelper(inorder, 0, inorder.size() - 1, postorder, 0,
                               postorder.size() - 1, inorderMap);
    }
};
```

THANK YOU !