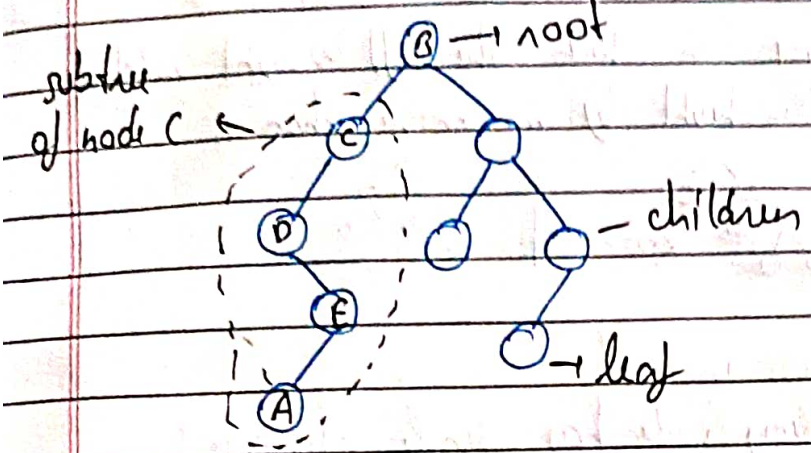
non-linear ← Tree → Hierarchy DS

Binary Tree:
- at most 2 children

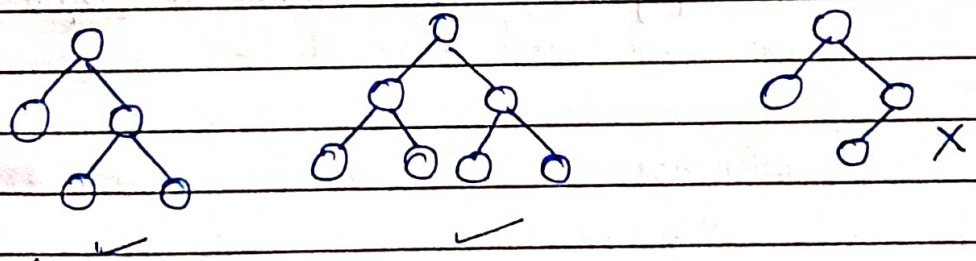
descendants:
- सभी node के नीचे के subtree descendants हैं।



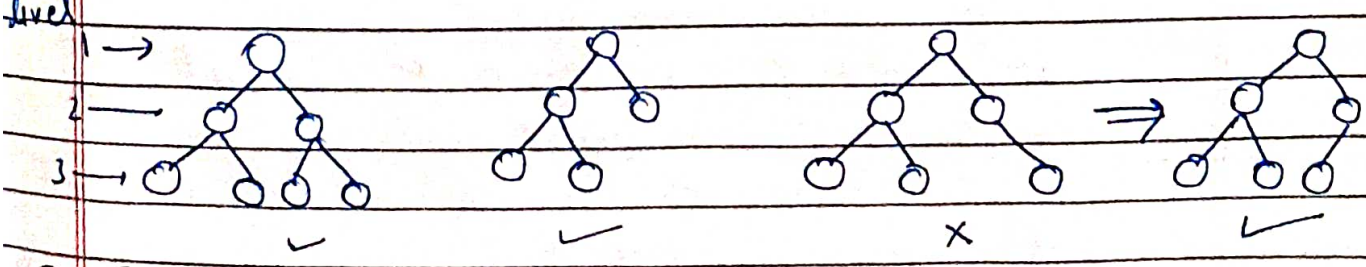
- for node A, (B, C, D, E) are ancestors

Types of Binary Tree:

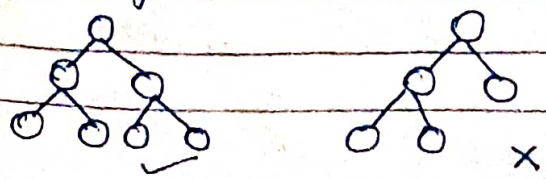
① Full Binary Tree:
either 0 or 2 children



② Complete BT
- all levels are completely filled except the last level
- the last level has all nodes on left as possible.



③ Perfect BT
- all leaf nodes are at same level

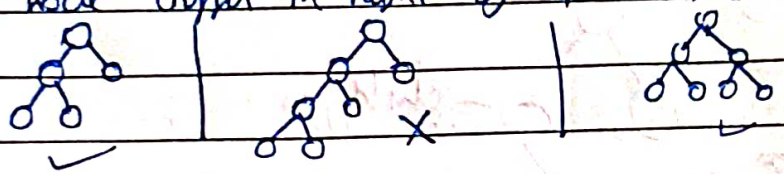


height of ~~tree~~ node

the no. of edges from the leaf node to the particular node ~~of~~ in the longest path.

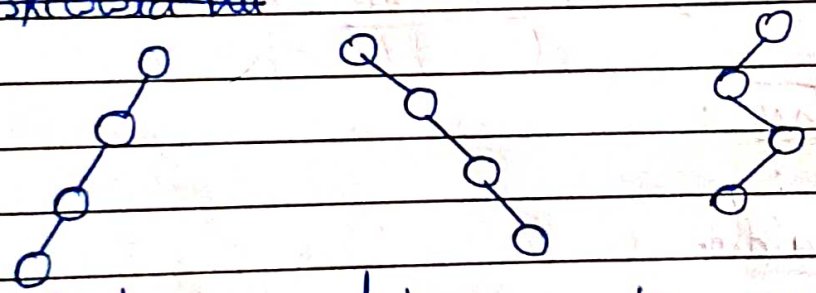
d) Balanced BT

- height of tree can be maximum $\log N$ (N = nodes)
- eg: $n = 8 \quad \log_2 8 = 3$
- A BT is Balanced BT in which the left & right subtrees of every node differ in height by no more than 1.



e) Degenerate Tree (every internal node has single children)

~~- skewed tree~~



- such tree are performance wise same as Linked list.

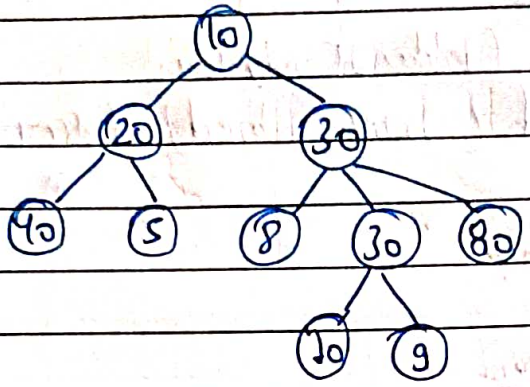
Application of Tree

1. Used to represent Hierarchical Data
   - Organization structure
   - Folder " (folder & subfolder)
   - XML/HTML content (JSON object) (HTML tag tree के form में होती है)
   - In OOP (Inheritance)
2. Binary Search Tree
3. Binary Heap (Used to represent Priority Queue)
   Used in Dijkstra, Prim, Huffman
4. B and B+ Trees in DBMS
   (B " " " are in DBMS for indexing)
5. Router use shortest path, so it build sp tree
6. Spanning & Shortest path tree in Computer Network
7. Parse Tree, Expression Tree in Compiler

Variation of Trees:

1. Trie: used to represent dictionary
   support operation like prefix search
2. Suffix Tree used to do fast search in String (pattern & Text)
3. Binary Index Tree: used for range Query Searches with range
4. Segment Tree:
   - more powerful
   - BIT is faster for limited set of operations.

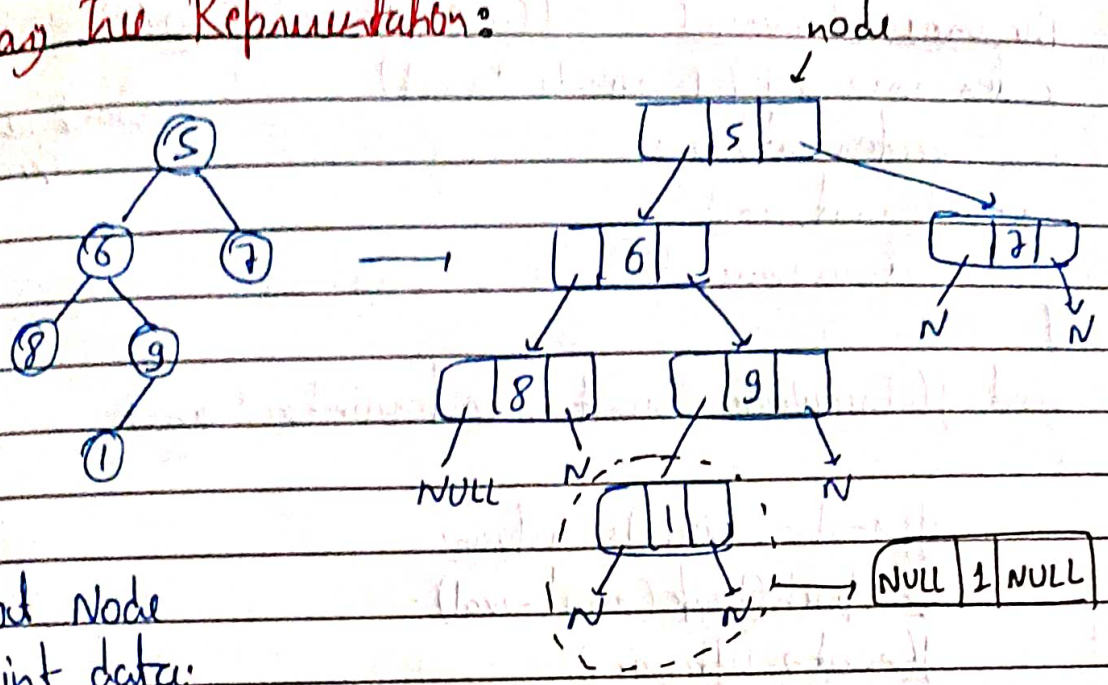
→ Degree of a node: no. of children it has
→ internal node: node which are not leaf are called IN



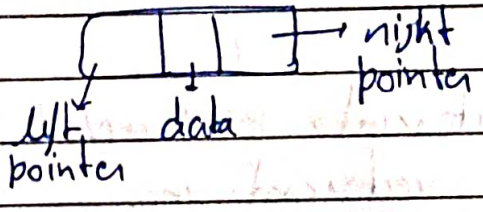
leaf node
LN: 40, 5, 8, 70, 9, 80
IN: 10, 20, 30, 30, 80

Important
① Binary Search Tree is the most used Tree DS
↓
it is a Binary Tree Variation
② Binary Heap Tree is used to represent Priority Queue
↓
it is a Binary Tree Variation
③ Segment Tree DS is also a Binary Tree.

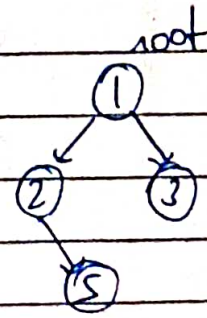
Binary Tree Representation:



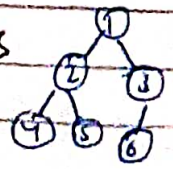

```
struct Node
{ int data;
  struct Node *left;
  ________ *right;
  Node(int val)
  { data = val;
    left = right = NULL;
  }
};
```




```
eg: main()
  { struct Node *root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->right = new Node(5);
  }
```



DFS
- Inorder (Left Root Right)
- Preorder (Root Left Right)
- Postorder (Left Right Root)

BFS  → 1 2 3 4 5 6 (level wise traversal)

(I) Pre-Order Traversal T.C → O(N) S.C → O(N)

↓ worst case when skewed Tree

Recursive

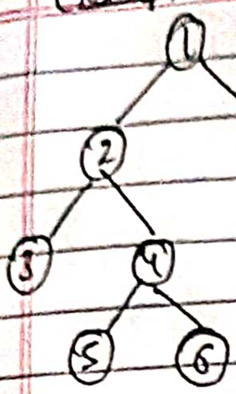
```
vector<int> PT(TreeNode *root)
{
    vector<int> ans;
    f(root, ans);
    return ans;
}
void f(TreeNode *root, vector<int> &ans)
{
    if(root == NULL) return;
    ans.push_back(root->val);
    f(root->left, ans);
    f(root->right, ans);
}
```

Iterative

```
vector<int> PT(TreeNode *root)
{
    vector<int> ans;
    if(root == NULL) return ans;
    stack<TreeNode *> st;
    st.push(root);
    while(!st.empty())
    {
        // TreeNode *root = st.top(); // TreeNode *node = st.top()
        st.pop();
        ans.push_back(node->val);
        if(node->right != NULL) st.push(node->right);
        if(node->left != NULL) st.push(node->left);
    }
    return ans;
}
```

Root L R

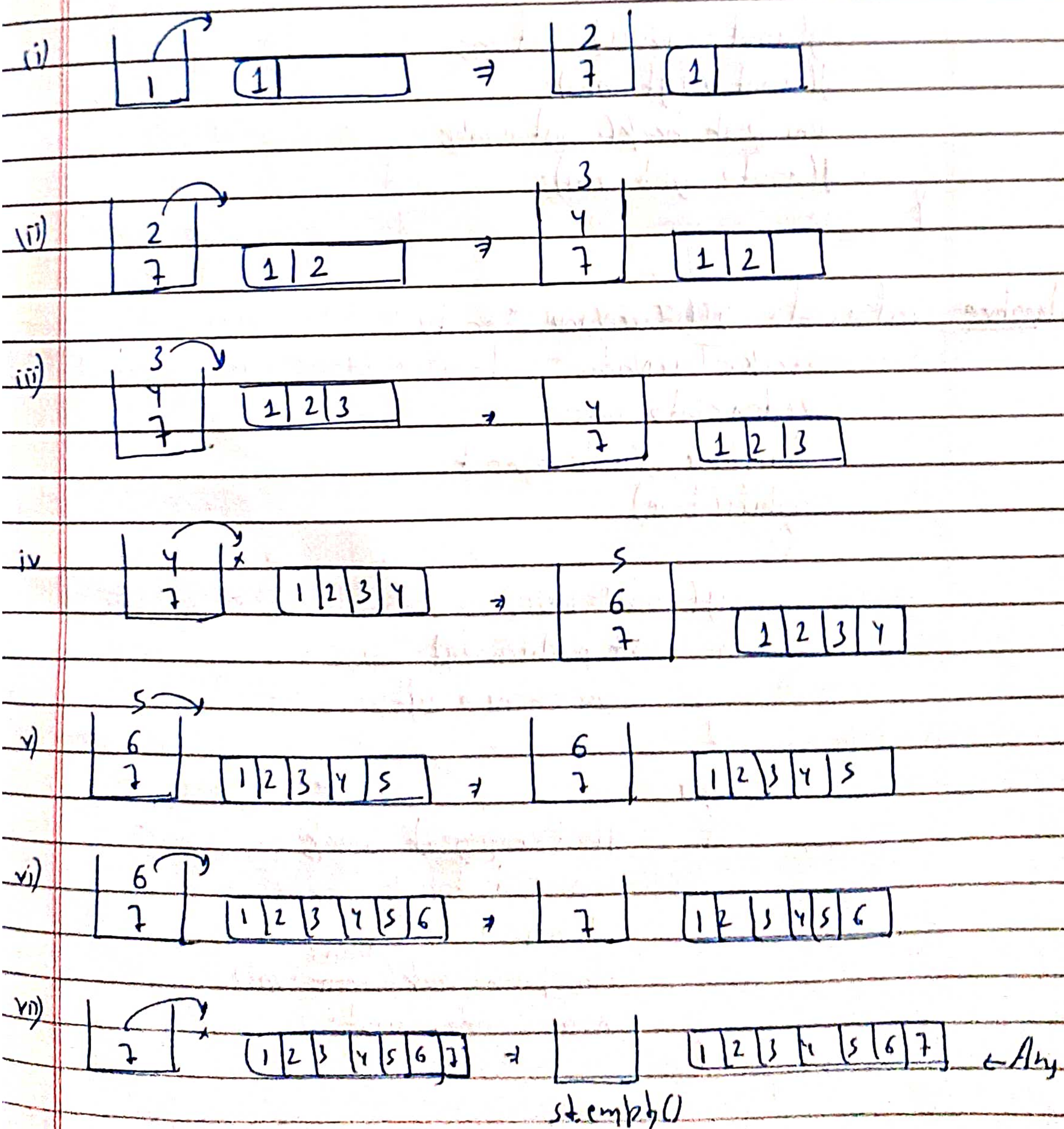
example



Ans - 1 2 3 4 5 6 7

Initially Stack [1] — 1 as a node समझना

पहले right वाला child stack में डालो क्योंकि जब निकाले तो left पहले निकले (Root L R) maintain रहे।



L Root R

(II) InOrder

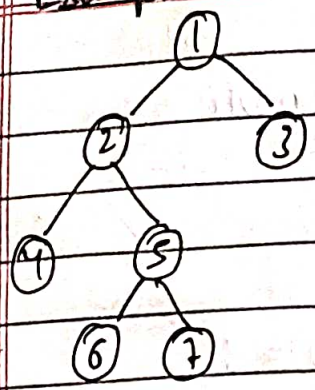
Recursive

```
vector<int> II(TreeNode * root)
{   vector<int> ans;
    f(root, ans);
    return ans;
}
void f(TreeNode *root, vector<int> &ans)
{
    if(root == NULL) return;
    f(root->left, ans);
    ans.push_back(root->val);
    f(root->right, ans);
}
```

Iterative

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

Example



Ans → 4 2 6 5 7 1 3

node = root

stack ⇒ visit node

1

(4 → left)

(i) node = node → left = NULL

4
2
1

⇒

stack ⇒ push node on left until (node != NULL)

(ii) node == NULL

node = 4

node = NULL

4
2
1

4		

iii) node == NULL

node = 2

node = 5

2
1

4	2	

iv) node != NULL

6
5
1

after 6 (6 → left = NULL)

node == NULL

⇒

6
5
1

4	2	6

node = 6

node = 6 → right = NULL

v)

5
1

4	2	6	5

node = 5

node = (5 → right) = 7

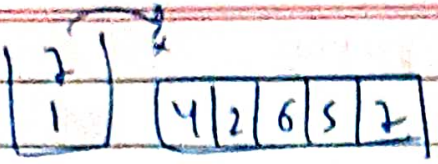
vi)

7
1

after 7 (7 → left == NULL)

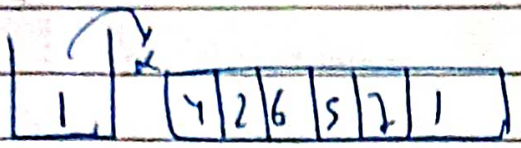
node = NULL

vii)



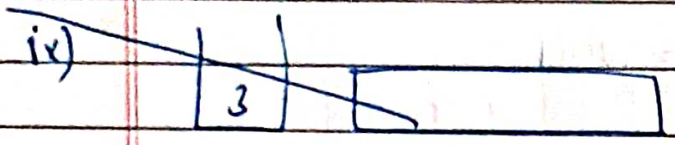
node = 7
node = (7 → right) = NULL

viii)

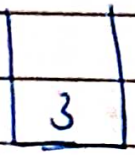


node = 1
node = (1 → right) = 3

ix)

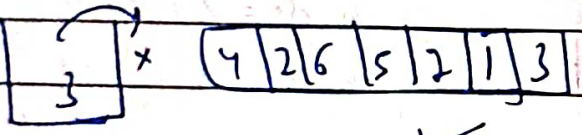


ix)



after 3 (3 → left = NULL)
node = NULL

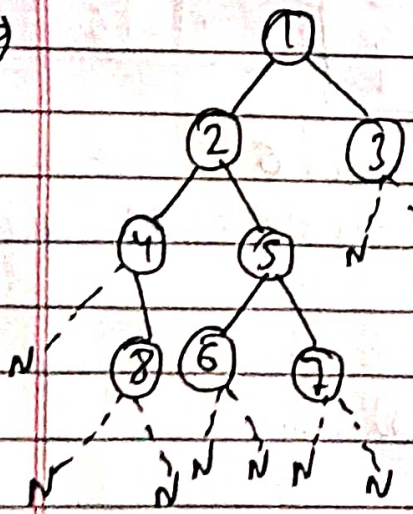
x)



node = 3
node = 3 → right = NULL

xi) st.empty() stop

⇒



L Root R
- 4 का left = NULL (ignore)
- 4 root पे print
- 4 के right पे जाओ, जाने के लिए node → right करो, क्योंकि st.top() वाली node है

4 show

L R Root

Post Order

Recursive

```
vector<int> PO(TreeNode *root)
{   vector<int> ans;
    f(root, ans);
    return ans;
}
void f(TreeNode *root, vector<int> &ans)
{
    if(root==NULL) return;
    f(node->left)
    f(node->right)
    ans.push_back(node->val);
}
```

a) Iterative O(N) SC→ O(2N)

in this sol^n you can simply use ans vector in place of st2. (then reverse vector to get ans)

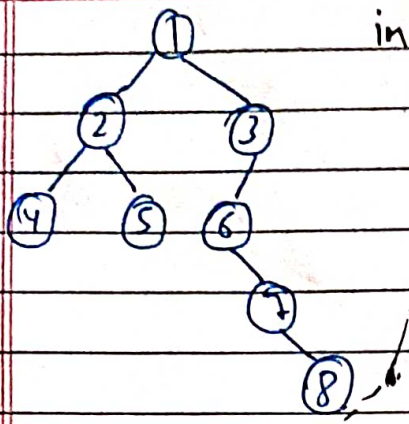
```
vector<int> PO(TreeNode *root)
{   vector<int> ans;
    if(root==NULL) return ans;
    stack<TreeNode *> st1, st2;
    st1.push(root);
    while(!st1.empty())
    {
        root = st1.top();
        st2.push(root);          // ans.push_back(root->val);
        st1.pop();
        if(root->left!=NULL) st1.push(root->left);
        if(root->right!=NULL) st1.push(root->right);
    }
    while(!st2.empty())
    {   ans.push_back(st2.top()->val);
        st2.pop();
    }
    return ans;
}
```

not required

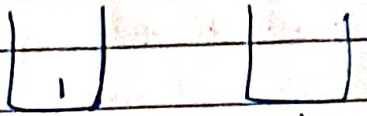
// reverse(ans);
returns ans;

L R Root

→



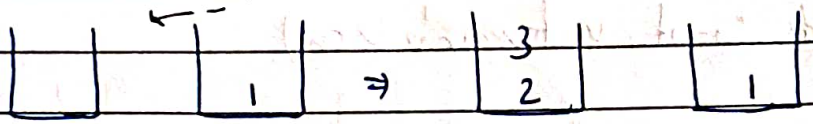
initial



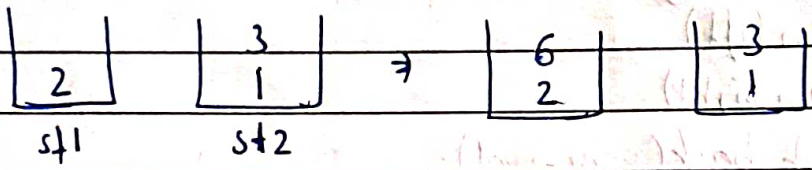
Ans 4 5 2 8 7 6 3 1

basically it is not maintain any 2 in st2 ...

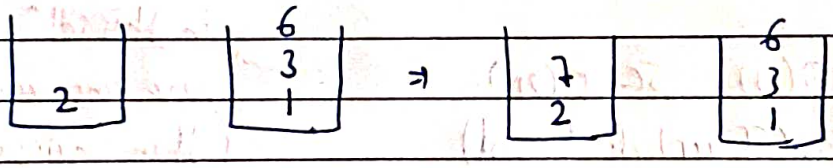
(i)



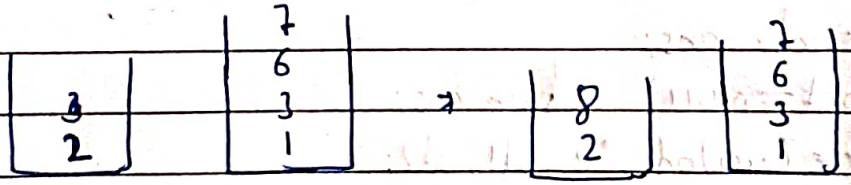
(ii)



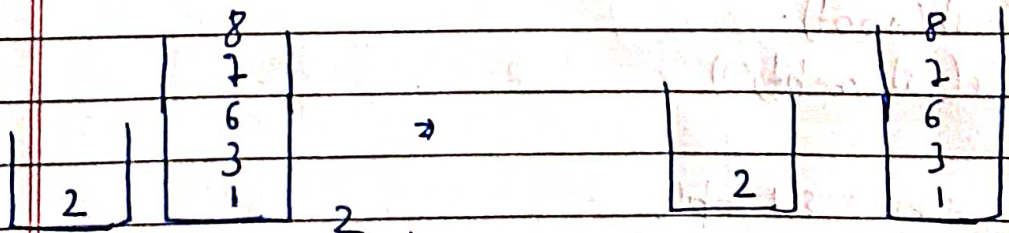
(iii)



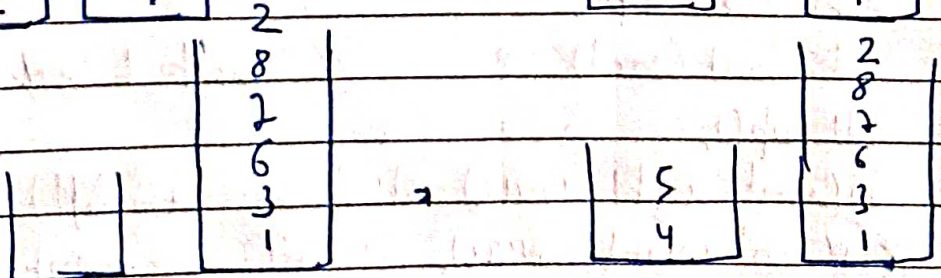
iv)



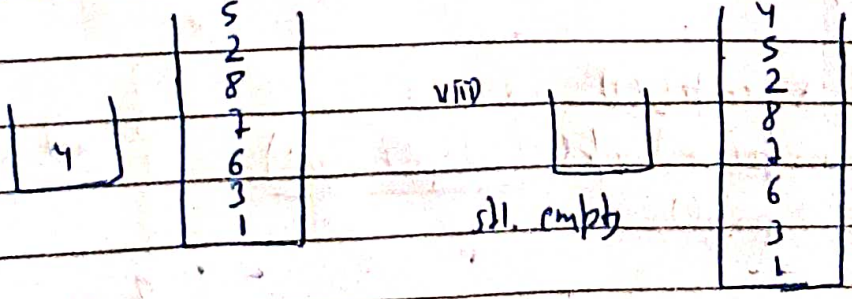
v)



vi)



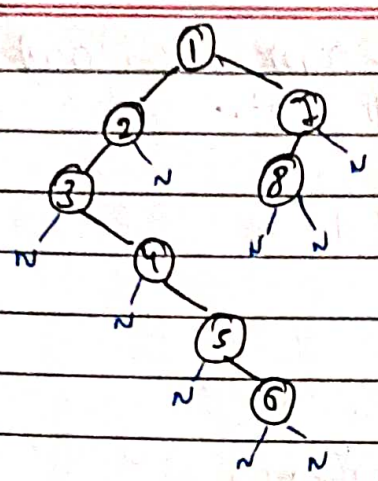
vii)



b) Iterative (with one Stack) TC- O(N) SC-: O(N)

```
vector<int> PO(TreeNode *root)
{   TreeNode *curr, *temp;
    stack<TreeNode *> st;
    vector<int> ans;
    if(root==NULL) return ans;
    curr=root;
    while(curr!=NULL || !st.empty())
    {
        if(curr!=NULL)
        {   st.push(curr);
            curr=curr->left;
        }
        else
        {
            temp=st.top()->right;
            if(temp==NULL)
            {   temp=st.top();
                st.pop();
                ans.push_back(temp->val);
                while(!st.empty() && temp==st.top()->right)
                {
                    temp=st.top();
                    st.pop();
                    ans.push_back(temp->val);
                }
            }
            else
                curr=temp;
        }
    }
    return ans;
}
```

Q)



curr = root = 1

st

left → left → left -- जाएंगे जब null हो जाएगा तब एक right जाएंगे फिर left → left → left -- जाएंगे।

~~ii)~~

i) curr = X X & N

3
2
1

ii) curr = N

temp = 3 → R = (4 != NULL)

curr = 4

iii) curr = 4 N

4
3
2
1

iv) curr = N

temp = 4 → R = (5 != NULL)

curr = 5

v) curr = 8 N

5
4
3
2
1

vi) curr = N

temp = 5 → R = (6 != NULL)

curr = 6

vii) curr = 6 N

6
5
4
3
2
1

viii) curr = N

temp = 6 → R = (NULL == NULL)

temp = 6

6 → x
5
4
3
2
1

6	

while loop चलेगा (temp == st.top() → right)

6 == (5 → R) → 6 == 6

temp = 5

5 == 4 → R → (5 == 5)

temp = 4

5 → x
4
3
2
1

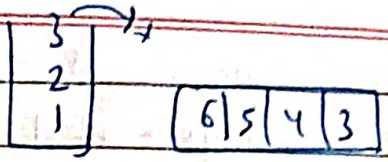
6	5	

4 → x
3
2
1

6	5	4

~~3 ==~~ Y == (3 → R) ⇒ Y == Y

temp = 3

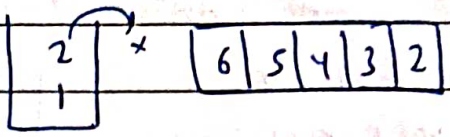


(3 == 2 → R) False while loop stop

ix) curr = N

temp = 2 → R = N

temp = 2



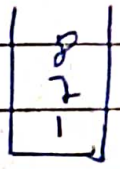
while loop start 2 == (1 → R) false stop

x) curr = N

temp = 1 → R = 7

~~temp~~ curr = 7

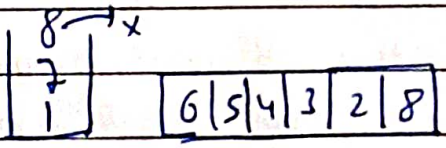
xi) curr = ~~7~~ ~~8~~ N



xii) curr = N

temp = 8 → R = N

temp = 8

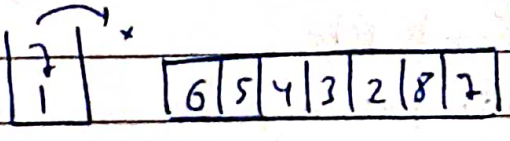


while loop start 8 == 7 → R false

xiii) curr = N

temp = 7 → R = N

~~curr~~ temp = 7



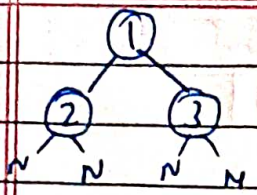
while loop start 7 == (1 → R) ⇒ (7 == 7)

temp = 1

st.empty()

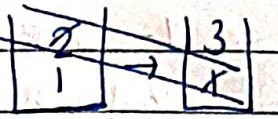
xiv) that while loop stops

eg

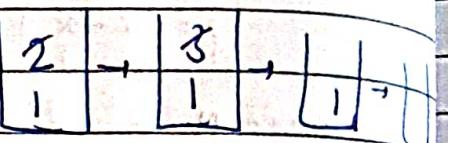


cum = 1 2 N 3 N

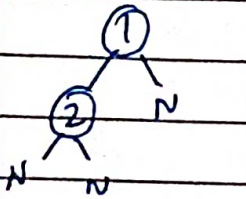
temp = ~~1~~ N 3 N 1



2 | 3 | 1



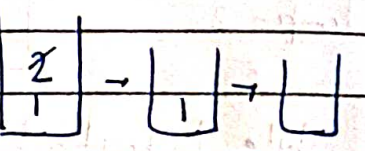
eg



1→R

cum = 1 2 N N

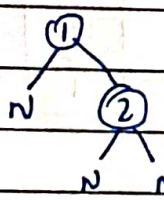
temp = N 2 N 1



2 | 1

stop

eg



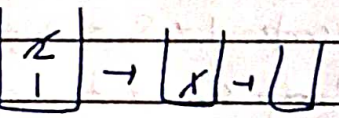
2→R

cum = 1 N 2 N N

temp = 2

i++ → 1→R

while loop



2 | 1

→ Preorder, Inorder, Postorder in One Traversal

```
vector<int> PIP(TreeNode * root)
{                                          T.C → O(3N)
    stack<pair<TreeNode *, int>> st;       O(4N)
    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);
}
}
```

Algo:

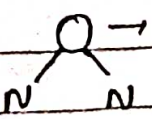
(node, num)

if num = 1 — Root push
push in preorder
num++
left push in stack
→ Root L R
→ Root के बाद left आयेगा ∴ left push

if num = 2
push in inorder
num++
right
→ L Root R

if num = 3
push in postorder
→ L R Root

* push करेगा root होगा ⁂

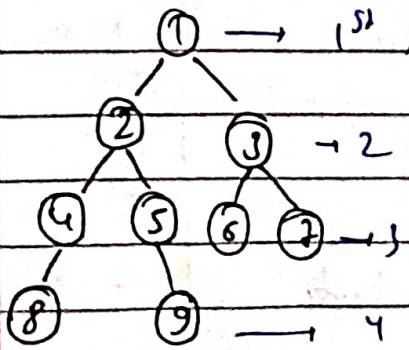


→ out as root (बाद left root ⁂ NULL आ)

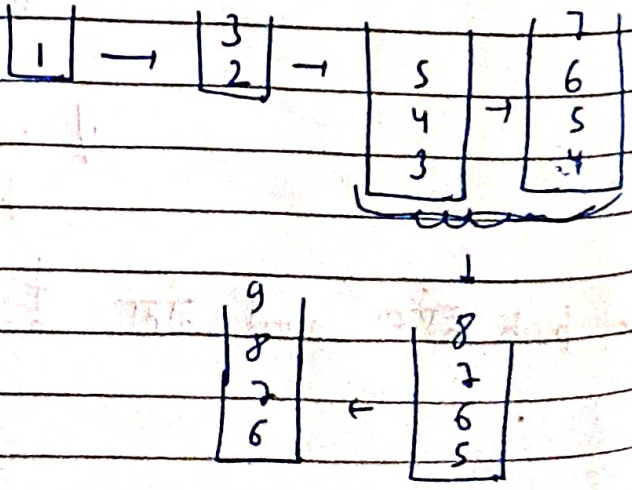
Level Order Traversal :

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

Logic



level wise queue ki pop diti !

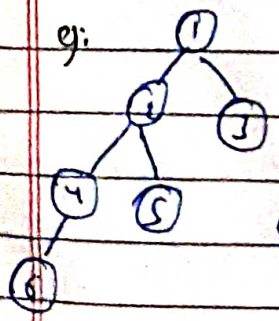


b)
```
void f(Node *root)
{
    if(root==NULL) return NULL;
    queue<Node *> q;
    q.push(root);
    while(!q.empty())
    {
        Node *cur = q.front();
        q.pop();
        cout<<(cur->val)<<" ";
        if(cur->left!=NULL) q.push(cur->left);
        if(cur->right!=NULL) q.push(cur->right);
    }
}
```

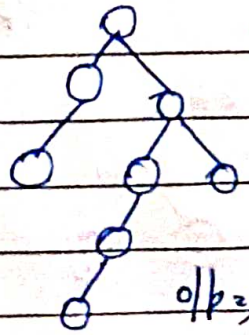
① Height of BT:

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

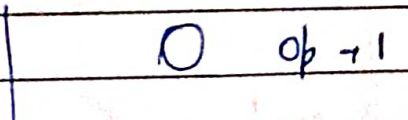
eg:



ob = 4

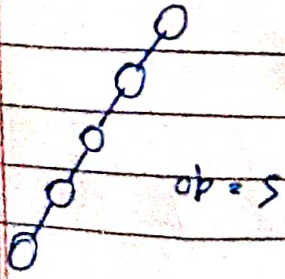


ob = 5

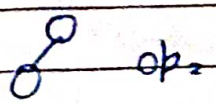


ob = 1

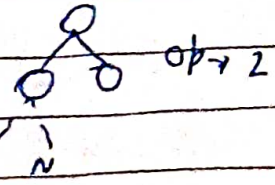
NULL ob = 0



ob = 5



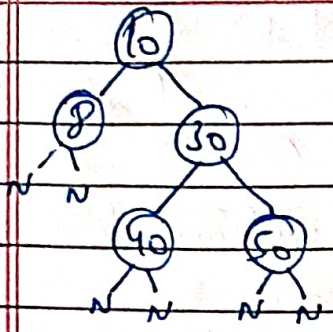
ob = 2

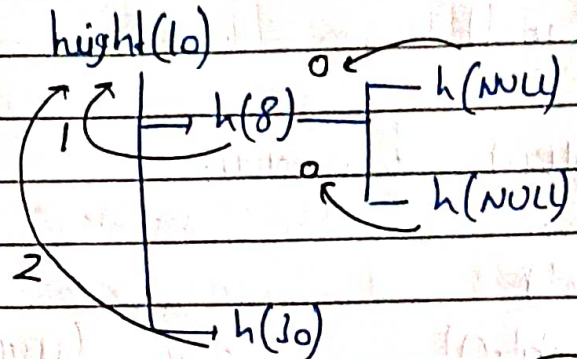


ob = 2

Classmate
Date
Page

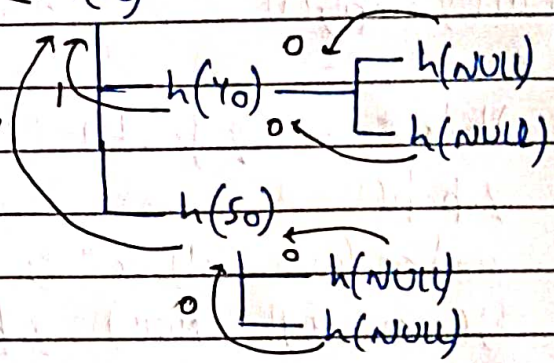
(I)





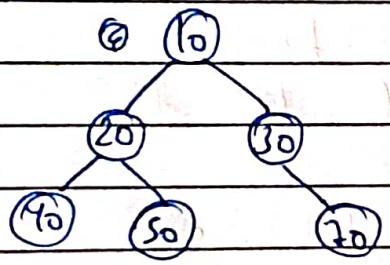
$\max(1,2)+1$

$2+1=3$ ✓



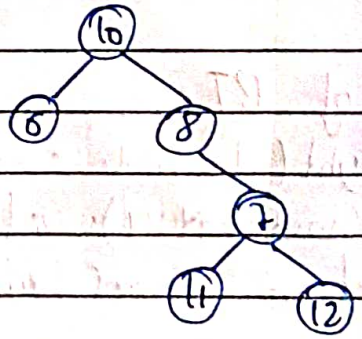
(II) **Print Nodes at distance k :**

$k=2$



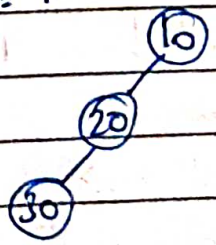
o/p → 40 50 70

$k=3$



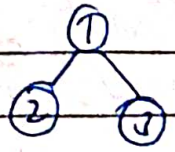
o/p 11 12

$k=1$



o/p → 20

$k=0$

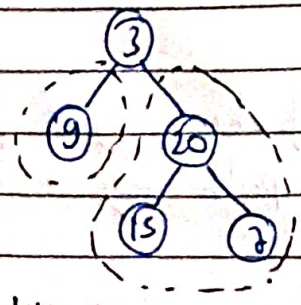


o/p = 1

```
vector<int> Kdistance(Node *root, int k)
{
    vector<int> ans;
    f(root, ans, k, 0);
    return ans;
}
void f(Node *root, vector<int> &ans, int k, int i)
{
    if(root==NULL) return;
    if(i==k)
    {
        ans.push_back(root->data);
        return;
    }
    f(root->left, ans, k, i+1);
    f(root->right, ans, k, i+1);
}
```

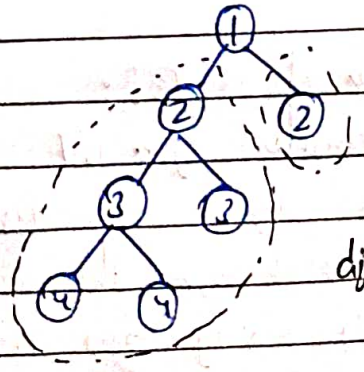
(III) **Balanced Binary Tree:**

A BT in which the left and right subtrees of **every node** differ in height by no more than 1.



differ by 1

ab+ 1 (True)



differ by 2

Logic for Balanced BT for every node;

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

a) Naive Calculate left height & right height of every node
$O(n^2)$

b) efficient $O(N)$

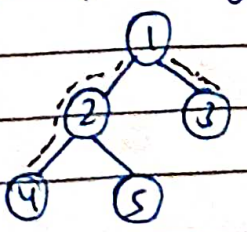
```
bool isBalanced(TreeNode *root)
{
    if(root==NULL) return 1;
    int ans = f(root);
    return ans==-1 ? 0 : 1;
}
int f(TreeNode *root)
{
    if(root==NULL) return 0;
    int h_left = f(root->left);
    if(h_left==-1) return -1;
    int h_right = f(root->right);
    if(h_right==-1) return -1;

    if(abs(h_left-h_right)>1) return -1;
    return max(h_left, h_right)+1;
}
```

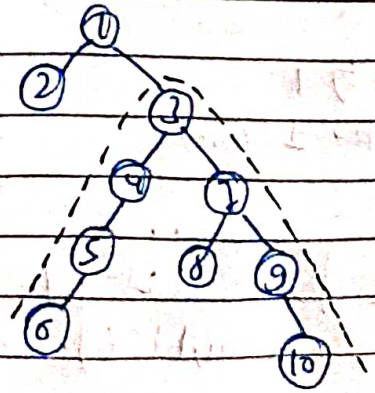
→ height of a tree question code

(IV) Diameter of a BT:

- The diameter of a BT is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.
- length is no. of edges b/w nodes

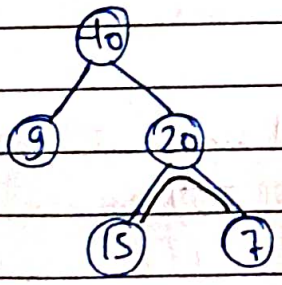


o/p = 3

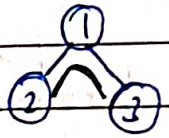


```
int diameter(TreeNode *root)
{   int ans;
    f(root, ans);
    return ans;
}
int f(TreeNode *root, int &ans)
{
    if(root==NULL) return 0;
    int h_left = f(root->left, ans);
    int h_right = f(root->right, ans);
    ans = max(ans, h_left + h_right);
    return max(h_left, h_right) + 1;
}
```

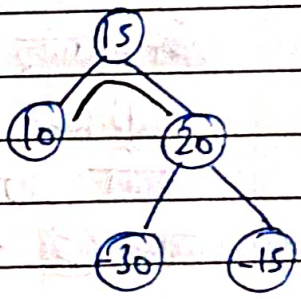
(V) BT Maximum Path Sum :



o/p 15+20+7 = 42



o/p = 6



o/p = 45

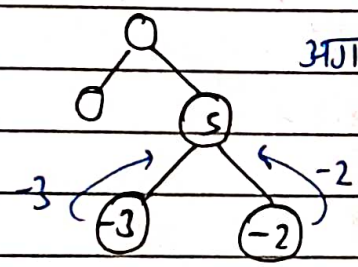
Note: Never include a -ve path in ans. (here -30 & -15 path is not included b/z they don't part in answer)

```
int maxPathSum(TreeNode *root)
{
    int ans = INT_MIN;
    f(root, ans);
    return ans;
}
```

```
int f(TreeNode *root, int &ans)
{
    if(root == NULL) return 0;
    int left_sum = max(0, f(root->left, ans));
    int right_sum = max(0, f(root->right, ans));

    ans = max(ans, root->val + left_sum + right_sum);
    return root->val + max(left_sum, right_sum);
}
```

o b/c -ve.

★ 

अगर left_sum = ~~max~~(f(root->left, ans) होता max वाला नहीं होता तो -3 return हो जाता left side से। और -2 return हो —— right —— और 5 को और -ve कर देता।

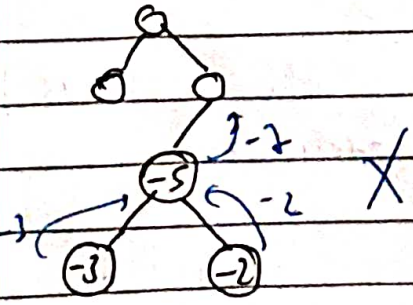
- ~~रहने~~ दीजिए

  left_sum = max(0, f(root->left, ans));

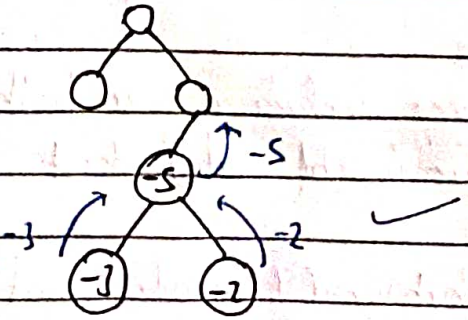
  ताकी अगर -ve है left_sum या right_sum we know it

- क्योंकि (5) है या (-5) हो तो -ve ही करेंगा।

Q.

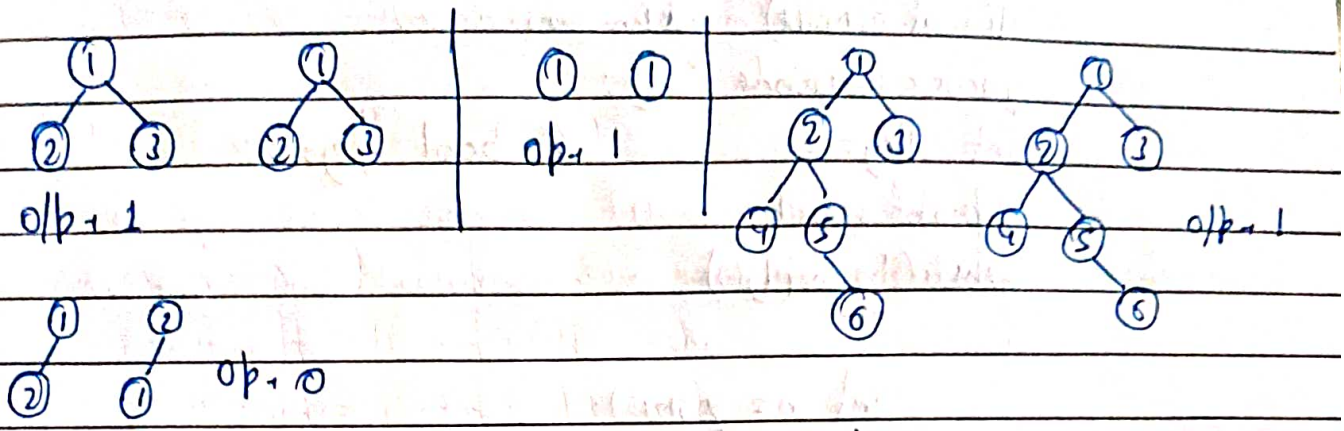


X



(VII) Same Tree:

2 BT are identical if they are structurally identical and the nodes have the same value.

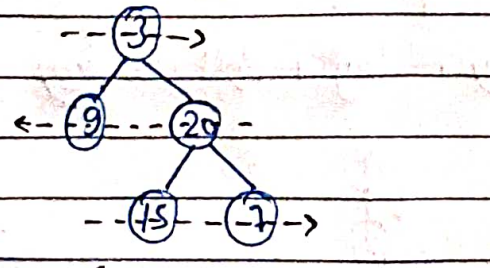



```
bool isSameTree(TreeNode *p, TreeNode *q)
{
    ★ if(p==NULL || q==NULL) return p==q;
      if(p->val == q->val)
          return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
      return 0;
}
```

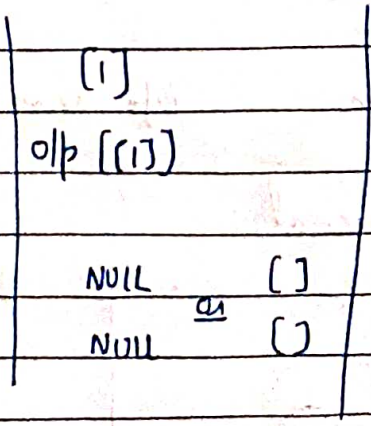
```
if(p==NULL && q==NULL) return 1;
if(p==NULL || q==NULL) return 0;
```

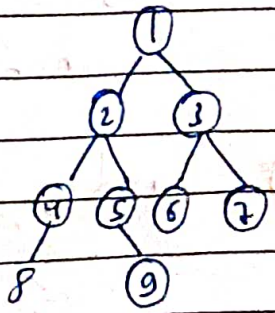
→ combinde of above line

(VIII) BT Zigzag Level Order Traversal:



o/p [[3], [20, 9], [15, 7]]



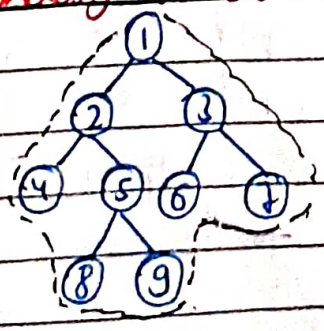


o/p [[1], [3,2], [4,5,6,7], [8,9]]

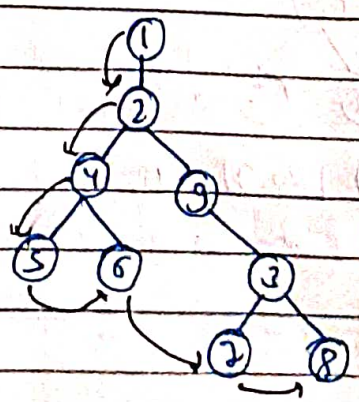
```
vector<vector<int>> zigzag(TreeNode *root)
{
    vector<vector<int>> ans;
    if(root==NULL) return ans;
    queue<TreeNode *> q;
    int flag=0;                    // bool flag=0;
    q.push(root);
    while(!q.empty())
    {
        int n=q.size();
        vector<int> v(n);
        for(int i=0;i<n;i++)
        {
            TreeNode *temp=q.front();
            q.pop();
            int ind=(flag==0)?i:n-i-1;
            v[ind]=temp->val;
            if(temp->left) q.push(temp->left);
            if(temp->right) q.push(temp->right);
        }
        ans.push_back(v);
        flag=(flag==0)?1:0;        // flag=!flag;
    }
    return ans;
}
```

Left boundary + leaf + Right Boundary

(VIII) Boundary Traversal:



op 1 2 4 8 9 6 7 3



T.C → O(N) + O(N) + O(N) → O(N)

S.C → O(N)

```
vector<int> boundary(Node *root)
{
    vector<int> v;
    if(root == NULL) return v;
    if(root->left == NULL && root->right == NULL)
    {
        v.push_back(root->data);
        return v;
    }
    v.push_back(root->data);  // put the root node already in v
    stack<int> st;
    Node *temp = root->left;
    while(temp)
    {
        if(temp->left == NULL && temp->right == NULL) break;
        v.push_back(temp->data);
        if(temp->left) == NULL) temp = temp->right;
        else temp = temp->left;
    }
    temp = root;
    inorder(root, v);
    temp = root->right;
    while(temp)
    {
        if(temp->left == NULL && temp->right == NULL) break;
        st.push(temp->data);
        if(temp->right == NULL) temp = temp->left;
        else temp = temp->right;
    }
    while(!st.empty())
    {
        v.push_back(st.top());
        st.pop();
    }
    return v;
}
```

edge case

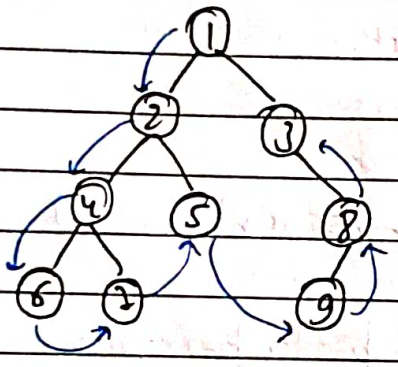
left boundary

inorder traversal for leaf node access

right boundary

```
void inorder(Node *root, vector<int> &v)
{
    if(root->left == NULL && root->right == NULL)
    {
        v.push_back(root->data);
        return;
    }
    if(root->left != NULL) inorder(root->left, v);
    if(root->right != NULL) inorder(root->right, v);
}
```

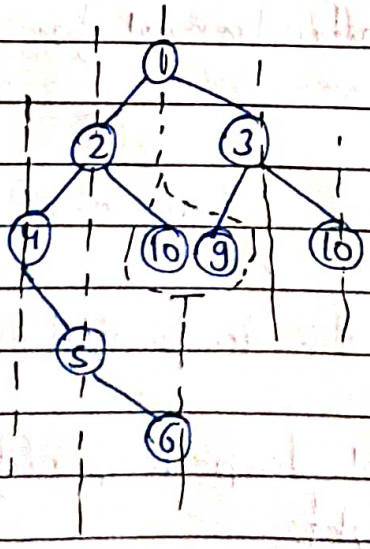
Example

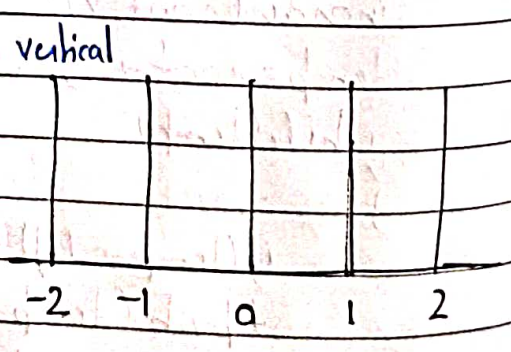


left boundary path ①, ②, ④ in vector v
inorder path ⑥, ⑦, ⑤, ⑨ ——
right boundary path ⑧, ③ ——

★★★

(IX) Vertical Order Traversal:





o/p [[4], [2,5], [1,6,9,10], [3], [10]]

↓

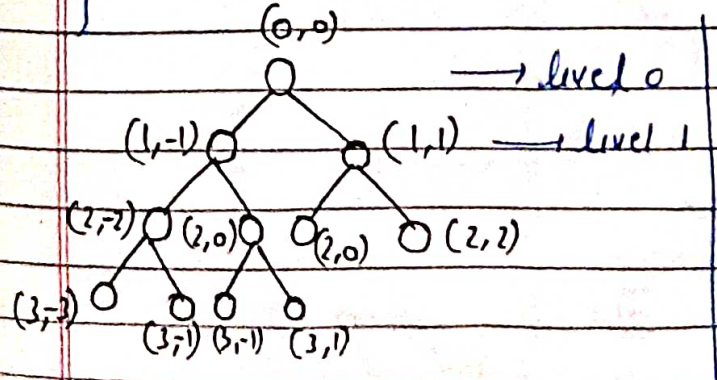
every vertical node in itself should be sorted

V.imp ★ एक level में कई nodes आपस में कहां एक same vertical में (like in above 10,9)

multiset uses .insert() function to insert

```
vector<vector<int>> vertical(TreeNode * root)
{
    queue<pair<TreeNode *, pair<int,int>>> q;
    map<int, map<int, multiset<int>>> m;
    q.push({root,{0,0}});
    while(!q.empty())
    {
        auto it = q.front();   beautiful line
        q.pop();
        TreeNode *node = it.first();
        int x=it.second.first, y=it.second.second;
        m[x][y].insert(node->val);
        if(node->left) q.push({node->left,{x-1,y+1}});
        if(node->right) q.push({node->right,{x+1,y+1}});
    }
    vector<vector<int>> ans;
    for(auto x:m)
    {
        vector<int> v;
        for(auto p:x.second)
            v.insert(v.end(), p.second.begin(), p.second.end());
        ans.push_back(v);
    }
    return ans;
}
```

x → vertical
y → level



(level, vertical)

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

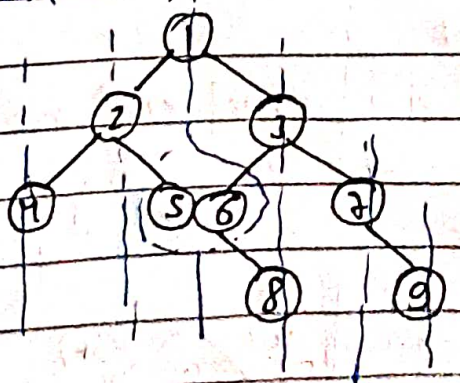
vertical — level — multiset b/z we want sorted vertical

left (-1,+1) right (+1,+1)

→ GFG Que (Vertical Traversal)

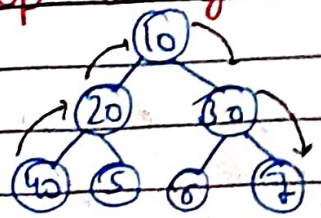
- If there are multiple nodes passing through a vertical line, then they should be printed as they appear in level order traversal of the Tree.

```
vector<int> VerticalOrder(Node *root)
{   queue<pair<Node*, int>> q;
    map<int, vector<int>> m;
    q.push({root, 0});
    while(!q.empty())
    {
        auto it = q.front();
        q.pop();
        Node *node = it.first;
        int x = it.second;
        m[x].push_back(node->data);
        if(node->left) q.push({node->left, x-1});
        if(node->right) q.push({node->right, x+1});
    }
    vector<int> ans;
    for(auto x:m)
    {   for(auto p: x.second)
            ans.push_back(p);
    }
    return ans;
}
```

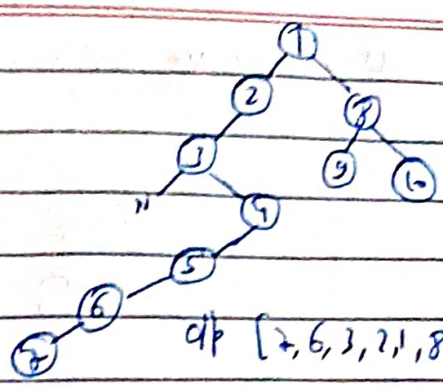


O/p 4 2 1 5 6 3 8 7 9.

(X) Top View of BT



o/p [40, 20, 10, 30, 7]



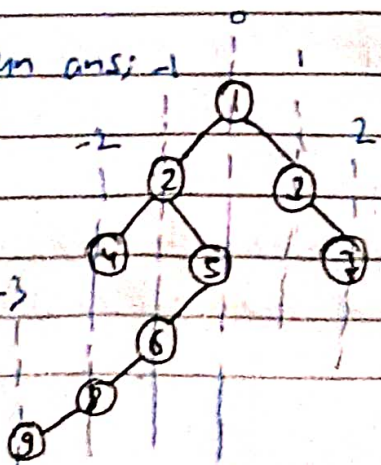
o/p [7, 6, 3, 2, 1, 8, 10]

```
vector<int> f(Node *root)
{
    vector<int> ans;
    if(root == NULL) return ans;
    queue<pair<Node*, int>> q;
    map<int, int> m;
    q.push({root, 0});
    while(!q.empty())
    {
        auto it = q.front();
        q.pop();
        Node *node = it.first;
        int x = it.second;
        if(m.find(x) == m.end()) m[x] = node->data;
        if(node->left) q.push({node->left, x-1});
        if(node->right) q.push({node->right, x+1});
    }
    for(auto x : m)
    {
        ans.push_back(x.second);
    }
    return ans;
}
```

* Using map bcz we want sort vertical

TC → O(N log N)

SC → O(N)

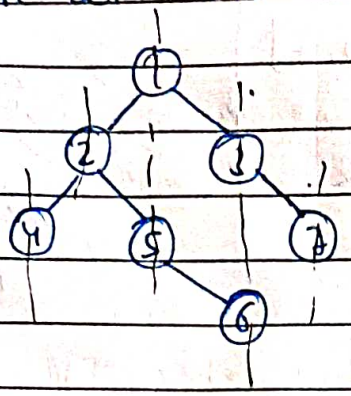


Queue

(node, vertical)

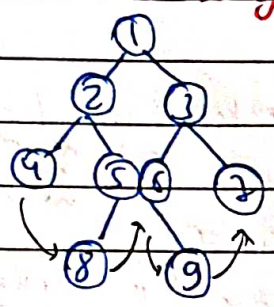
logic → simply vertical dir top Node पहचान लो

Recursive Sol^{n} DFS can't work b/z

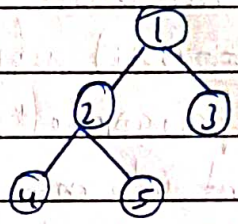


when iterating left we would have got (6) & inserted in map causing (3) ~~to~~ not to be inserted

(XI) Bottom View of BT



o/p : [4, 8, 6, 9, 7]

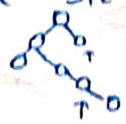


o/p [4, 2, 5, 3]

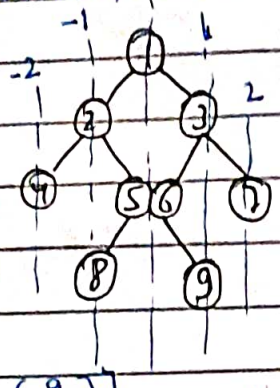
★ when nodes got overlap o/p the right node (here in the ⑤, ⑥ we o/p ⑥)

```
vector<int> f(Node *root)
{   vector<int> ans;
    if(root==NULL) return ans;
    queue<pair<Node*, int>> q;          TC - O(nlogn)
    map<int,int> m;                     SC - O(n)
    q.push({root,0});
    while(!q.empty())
    {
        auto it = q.front();
        q.pop();
        Node *node = it.first;
        int x = it.second;
        m[x] = node->data;
        if(node->left) q.push({node->left, x-1});
        if(node->right) q.push({node->right, x+1});
    }
```

Recursive soln not possible b/z map updates value. (see by eg):



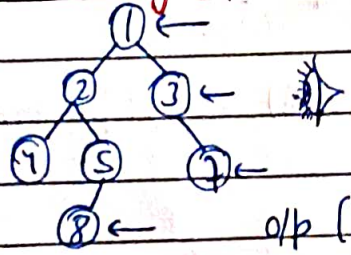
for(auto x:m)
    ans.push_back(x.second);
return ans;
}



o/p [4,8,6,9,7]

logic simply bottom node ka vertical dist ans me

(XII) Right Side View of BT



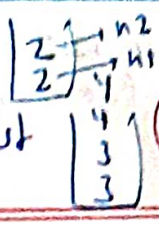
o/p [1,3,7,8]

```
vector<int> f(TreeNode * root)
{   vector<int> ans;
    f(root, ans, 0);
    return ans;
}
void f(TreeNode *root, vector<int> &v, int l)   // l -> level
{   if(root==NULL) return;
    if(v.size()==l)
        v.push_back(root->val);
    f(root->right, v, l+1);
    f(root->left, v, l+1);
}
```

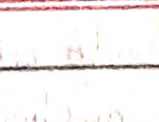
for left side view of BT ↓ swap these two line

logic ki level ki right element ans me push by maintaining l variable

initial stack



after 1st iteration of

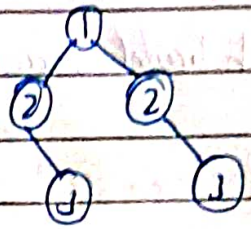


can be done with stack

discuss

Date
Page

(VIII) Symmetric Tree



mirror image

o/p → 0

o/p → 1

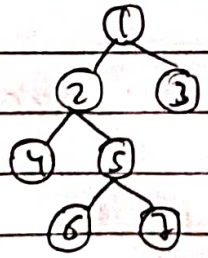
```
bool f(TreeNode *root)
{   return f(root, root);
}
bool f(TreeNode *p, TreeNode *q)
{
    if(p==NULL || q==NULL) (p==q);
    if (p->val == q->val)
        return f(p->left, q->right) && f(p->right, q->left);
    return 0;
}
```

O(N)

O(N) → D

log N

(VIV) Root to Node Path



Given n = 7

o/p [1, 2, 5, 7]

- No two nodes in tree are same
- Assume n to be present in tree.

```
vector<int> Path(TreeNode *root, int n)
{
    vector<int> ans;
    if(root==NULL) return ans;
    bool a = f(root, ans, n)
    return ans;
}
bool f(TreeNode *root, vector<int> &ans, int n)
{
```

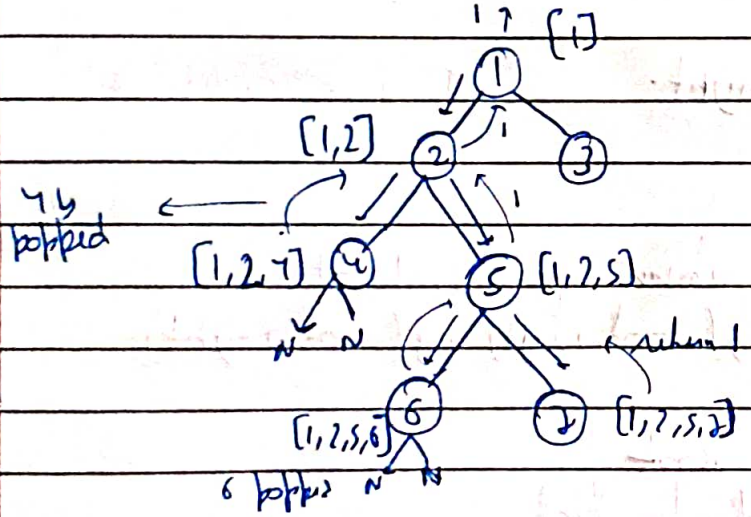
O(N)

S(+ O(H)

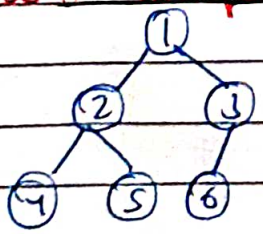
```
if(root==NULL) return 0;
ans.push_back(root->val);
if(root->val==n) return 1;

if(f(root->left, ans, n) || f(root->right, ans, n))
    return 1;
ans.pop_back(); // when backtracking pop-back
return 0;
}
```

if n return 1 to caller, caller return 1 to caller so on.



(XV) Count Complete Tree Nodes



Given Complete BT

o/p → 6

[]	[1]
o/p → 0	o/p → 1

T.C → O(N)

```
int countNodes(TreeNode* root)
{   int c=0;
    f(root, c);
    return c;
}
```

```
void f(TreeNode *root, int &c)
{   if(root==NULL) return;
    c++;
    f(root->left, c);
    f(root->right, c);
}
```

Crazy sol^{n} see the implementation

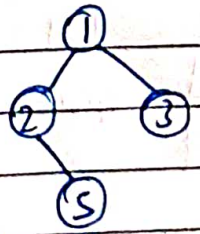
```
int f(TreeNode *root)
{
    if(root==NULL) return 0;
    TreeNode *l=root, *r=root;
    int lh=0, rh=0;
    while(l)
    {   lh++;
        l=l->left;
    }
    while(r)
    {   rh++;
        r=r->right;
    }
    if(lh==rh) return pow(2,lh)-1;
    return 1+f(root->left)+f(root->right);
}
```

TC → $(\log n)^2$

(XVI) Binary Tree All paths to leaf

Root

return all root to leaf path



op ["1->2->5","1->3"]

```
vector<string> f1(TreeNode *root)
{
    vector<string> ans;
    string s;
    f(root, ans, s);
    return ans;
}
```

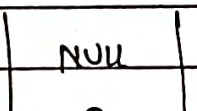
```
void f(TreeNode *root, vector<string> &ans, string s)
{
    if(root == NULL) return;
    if(root->left == NULL && root->right == NULL)
    {
        s += to_string(root->val);
        ans.push_back(s);
        return;
    }
    f(root->left, ans, s + to_string(root->val) + "->");
    f(root->right, ans, s + to_string(root->val) + "->");
}
```

branch of Recursion

Possibly any problem in while backtracking.

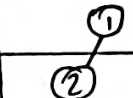
(XVII) ~~Size~~ No. of nodes of BT

o/p → 5

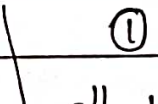


NULL

o



o/p → 3



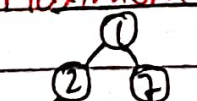
o/p → 1

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

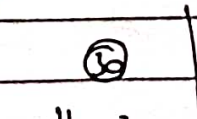
O(n)

S.C. → O(h)

(XVIII) Maximum in a BT



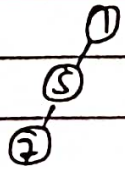
o/p → 9



-30

NULL

INT_MIN



o/p → 7

```
int f(TreeNode *root)
{   if(root == NULL)
        return INT_MIN;
    return max(root->val, max(f(root->left), f(root->right)));
}
```