BINARY SEARCH TREE

DEF:

LEFT CHILD LESSER THAN ROOT  
GREATER THAN RIGHT

TC FOR BT-O(N)

TC FOR BST-O(LOG N)

**1)FROM SCRATCH INSERTING A NODE**

struct Node {

int value;

Node\* left;

Node\* right;

Node(int val) : value(val), left(nullptr), right(nullptr) {}

};

class Solution {

public:

TreeNode\* insertIntoBST(TreeNode\* root, int val) {

if(root==NULL) return new TreeNode(val);

TreeNode\* curr=root;

while(true){

if(curr->val<val){

if(curr->right!=NULL){

curr=curr->right;

}

else{

curr->right=new TreeNode(val);

break;

}

}

else{

if(curr->left!=NULL){

curr=curr->left;

}

else{

curr->left=new TreeNode(val);

break;

}

}

}

return root;

}

};

2)VALIDATE BST

**class Solution {**

**public:**

**TreeNode\* prev=nullptr;**

**void inorder(TreeNode\* &root,bool &flag){**

**if(root==NULL || !flag) return;**

**inorder(root->left,flag);**

**if(prev && prev->val>=root->val){**

**flag=false;**

**return;**

**}**

**prev=root;**

**inorder(root->right,flag);**

**}**

**bool isValidBST(TreeNode\* root) {**

**bool flag=true;**

**inorder(root,flag);**

**return flag;**

**}**

**};**

**3)TOP VIEW**

**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)**

**{**

**//Your code here**

**vector<int>ans;**

**if(root==NULL) return ans;**

**queue<pair<Node\*,int>>q;**

**map<int,int>mpp;**

**q.push({root,0});**

**while(!q.empty()){**

**auto it=q.front();**

**q.pop();**

**Node\* node=it.first;**

**int line=it.second;**

**if(mpp.find(line)==mpp.end()){**

**mpp[line]=node->data;**

**}**

**if(node->left!=NULL) {**

**q.push({node->left,line-1});**

**}**

**if(node->right!=NULL){**

**q.push({node->right,line+1});**

**}**

**}**

**for(auto it:mpp){**

**ans.push\_back(it.second);**

**}**

**return ans;**

**}**

**};**

**4)BOTTOM VIEW**

class Solution {

public:

vector <int> bottomView(Node \*root) {

// Your Code Here

vector<int>ans;

if(root==NULL) return ans;

queue<pair<Node\*,int>>q;

map<int,int>mpp;

q.push({root,0});

while(!q.empty()){

auto it=q.front();

q.pop();

Node\* node=it.first;

int line=it.second;

mpp[line]=node->data;

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

if(node->right!=NULL) q.push({node->right,line+1});

}

for(auto it:mpp){

ans.push\_back(it.second);

}

return ans;

}  
6)RIGHT VIEW

class Solution {

public:

    vector<int> rightSideView(TreeNode\* root) {

        vector<int>ans;

        findrighview(root,0,ans);

        return ans;

    }

    void findrighview( TreeNode\* root,int level,vector<int>&ans){

        if(root==NULL) return;

        if(ans.size()==level){

            ans.push\_back(root->val);

        }

        findrighview(root->right,level+1,ans);

        findrighview(root->left,level+1,ans);

    }

7)LEFT VIEW

vector<int> leftView(Node \*root) {

// code here

vector<int>ans;

findleftview(root,0,ans);

return ans;

}

void findleftview( Node\* root,int level,vector<int>&ans){

if(root==NULL) return;

if(ans.size()==level){

ans.push\_back(root->data);

}

findleftview(root->left,level+1,ans);

findleftview(root->right,level+1,ans);

}