**1.VALIDATE BST**  
  
class Solution {

public:

    void inorder\_traverse(TreeNode\* root,vector<int>&nodes)

    {

        if(!root) return;

        inorder\_traverse(root->left,nodes);

        nodes.push\_back(root->val);

        inorder\_traverse(root->right,nodes);

    }

    bool isValidBST(TreeNode\* root) {

         if(!root) return true;

         vector<int>nodes;

         inorder\_traverse(root,nodes);

         set<int>s(nodes.begin(),nodes.end());

         if(s.size()!=nodes.size())

         return false;

         for(int i=0;i<nodes.size()-1;i++){

            if(nodes[i]>nodes[i+1])

            return false;

         }

         return true;

    }

};

**2.If not BST, convert to BST**  
  
class Solution {

public:

void inorder\_traverse(TreeNode\* root, vector<int>& nodes) {

if (!root) return;

inorder\_traverse(root->left, nodes);

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

inorder\_traverse(root->right, nodes);

}

void assign\_sorted\_values(TreeNode\* root, vector<int>& sorted\_values, int& index) {

if (!root) return;

assign\_sorted\_values(root->left, sorted\_values, index);

root->val = sorted\_values[index++];

assign\_sorted\_values(root->right, sorted\_values, index);

}

TreeNode\* convertToBST(TreeNode\* root) {

if (!root) return nullptr;

vector<int> nodes;

inorder\_traverse(root, nodes);

sort(nodes.begin(), nodes.end());

int index = 0;

assign\_sorted\_values(root, nodes, index);

return root;

}

};

**3.TOP VIEW**  
  
class Solution {

public:

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

// code here

if(!root)

return{};

map<int,int>hd\_map;

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

q.push({root,0});

while(!q.empty()){

Node\* node = q.front().first;

int hd = q.front().second;

q.pop();

if(hd\_map.find(hd)==hd\_map.end()){

hd\_map[hd]=node->data;

}

if(node->left){

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

}

if(node->right){

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

}

}

vector<int>res;

for (auto it : hd\_map) {

res.push\_back(it.second);

}

return res;

}

};

**4. BOTTOM VIEW**  
  
class Solution {

public:

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

if (!root) return {};

map<int, int> hd\_map;

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

q.push({root, 0});

while (!q.empty()) {

Node\* node=q.front().first;

int hd=q.front().second;

q.pop();

hd\_map[hd] = node->data;

if (node->left) {

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

}

if (node->right) {

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

}

}

vector<int> res;

for (auto i : hd\_map) {

res.push\_back(i.second);

}

return res;

}

};  
  
  
**5. LEFT VIEW**  
  
class Solution {

public:

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

if(!root)return{};

vector<int>res;

queue<Node\*>q;

q.push(root);

while(!q.empty()){

int level\_size=q.size();

for(int i=0;i<level\_size;i++){

Node\* node=q.front();

q.pop();

if(i==0){

res.push\_back(node->data);

}

if(node->left)

q.push(node->left);

if(node->right)

q.push(node->right);

}

}

return res;

}

};  
  
  
**6. RIGHT VIEW**

class Solution

{

public:

vector<int> rightView(Node \*root)

{

// Your Code here

if(!root)return{};

vector<int>res;

queue<Node\*>q;

q.push(root);

while(!q.empty()){

int level\_size=q.size();

for(int i=0;i<level\_size;i++){

Node\* node=q.front();

q.pop();

if(i==level\_size-1){

res.push\_back(node->data);

}

if(node->left)

q.push(node->left);

if(node->right)

q.push(node->right);

}

}

return res;

}

};