**1. Level order traversing**

**vector<vector<int>> res;**

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

**queue<TreeNode\*> que;**

**que.push(root);**

**while(!que.empty())**

**{**

**vector<int> level;**

**int n = que.size();**

**for(int i=0;i<n;i++)**

**{**

**TreeNode\* t = que.front(); que.pop();**

**level.push\_back(t->val);**

**if(t->left)**

**que.push(t->left);**

**if(t->right)**

**que.push(t->right);**

**}**

**res.push\_back(level);**

**}**

**return res;**

**2. Reverse Level Order traversal**

**Just reverse the ‘res’ either by STL ‘reverse’ or using stack.**

**Store each level in stack, then after the queue is empty, pop each level from stack and push\_back to res.**

**3. Maximum depth of a tree**

**int maxDepth(TreeNode\* root) {**

**if(root == nullptr)**

**return 0;**

**int leftmax = maxDepth(root->left);**

**int rightmax = maxDepth(root->right);**

**return 1+max(leftmax,rightmax);**

**}**

**4. Diameter of Tree**

**int diameterOfBinaryTree(TreeNode\* root) {**

**int d=0;**

**dfs(root,d);**

**return d;**

**}**

**int dfs(TreeNode\* root,int&d)**

**{**

**if(root == nullptr)**

**return 0;**

**int left = dfs(root->left,d);**

**int right = dfs(root->right,d);**

**d = max(d,left+right);**

**return 1+max(left,right);**

**}**

**5. Mirror a binary tree**

**TreeNode\* invertTree(TreeNode\* root) {**

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

**TreeNode\* left = invertTree(root->left);**

**TreeNode\* right = invertTree(root->right);**

**root->left = right;**

**root->right = left;**

**return root;**

**}**

**6. Binary tree inorder traversal**

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

**stack<TreeNode\*> stk;**

**vector<int> res;**

**while(true)**

**{**

**if(root != nullptr)**

**{**

**stk.push(root);**

**root = root->left;**

**}**

**else**

**{**

**if(stk.empty()) break;**

**res.push\_back(stk.top()->val);**

**root = stk.top()->right;**

**stk.pop();**

**}**

**}**

**return res;**

**}**

**7. Binary tree preorder traversa;**

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

**if(root == nullptr)**

**return vector<int>();**

**vector<int> res;**

**stack<TreeNode\*> stk;**

**stk.push(root);**

**while(!stk.empty())**

**{**

**TreeNode \*curr = stk.top();**

**res.push\_back(curr->val);**

**stk.pop();**

**if(curr->right)**

**stk.push(curr->right);**

**if(curr->left)**

**stk.push(curr->left);**

**}**

**return res;**

**}**

**8. Binary tree postorder traversal**

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

**if(root == nullptr) return vector<int>();**

**vector<int> res;**

**stack<TreeNode\*> stk;**

**TreeNode \*last = NULL;**

**while(!stk.empty() || root !=nullptr)**

**{**

**if(root)**

**{**

**stk.push(root);**

**root = root->left;**

**}**

**else**

**{**

**TreeNode \*node = stk.top();**

**if(node->right && last!=node->right)**

**root = node->right;**

**else**

**{**

**res.push\_back(node->val);**

**last = node;**

**stk.pop();**

**}**

**}**

**}**

**return res;**

**}**

**9. Left side view**

**10. Right side view**

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

**if(root == NULL) return vector<int>();**

**vector<int> RightSide;**

**vector<int> LeftSide;**

**queue<TreeNode\*> q;**

**q.push(root);**

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

**{**

**vector<int> level;**

**int size = q.size();**

**for(int i=0;i<size;i++)**

**{**

**TreeNode \*curr = q.front(); q.pop();**

**level.push\_back(curr->val);**

**if(curr->left) q.push(curr->left);**

**if(curr->right) q.push(curr->right);**

**}**

**Leftside.push\_back(level[0]);**

**RightSide.push\_back(level[level.size()-1]);**

**}**

**return LeftSide or return RightSide;**

**}**