	Problem	Solution	Time Complexity	Failure case
1 13/01/20	Clone a Binary Tree with Random Pointers	This problem is similar to clone a LL with random pointers. Method 1: 1) Recursively traverse the given Binary and copy key value, left pointer and right pointer to clone tree. While copying, store the mapping from given tree node to clone tree node in a hashtable. In the following pseudo code, 'cloneNode' is currently visited node of clone tree and 'treeNode' is currently visited node of given tree. cloneNode->key = treeNode->key cloneNode->left = treeNode->left cloneNode->right = treeNode->right map[treeNode] = cloneNode 2) Recursively traverse both trees and set random pointers using entries from hash table. cloneNode->random = map[treeNode->random] Method 2: 1. Create new nodes in cloned tree and insert each new node in original tree between the left pointer edge of corresponding node in the original tree (See the below image). i.e. if current node is A and it's left child is B (A — >> B), then new cloned node with key A wil be created (say cA) and it will be put as A — >> cA — >> B (B can be a NULL or a non-NULL left child). Right child pointer will be set correctly i.e. if for current node A, right child is C in original tree (A — >> C) then corresponding cloned nodes cA and cC will like cA —- >> cC 2. Set random pointer in cloned tree as per original tree i.e. if node A's random pointer points to node B, then in cloned tree, cA will point to cB (cA and cB are new node in cloned tree corresponding to node A and B in original tree) 3. Restore left pointers correctly in both original and cloned tree		
2	Write a Program to Find the Maximum	1. If tree is empty then return 0 2. Else	O(n)	

	Depth or Height of a Tree	(a) Get the max depth of left subtree recursively i.e., call maxDepth(tree->left-subtree) (a) Get the max depth of right subtree recursively i.e., call maxDepth(tree->right-subtree) (c) Get the max of max depths of left and right subtrees and add 1 to it for the current node. max_depth = max(max dept of left subtree, max depth of right subtree) + 1 (d) Return max_depth		
3	Count Leaves in Binary Tree	Method 1: Use level order traversal (queue) Check node whose left and right child are null, increment count. Method 2: Use recursion, check if left and right child are null and increment 1) If node is NULL then return 0. 2) Else If left and right child nodes are NULL return 1. 3) Else recursively calculate leaf count of the tree using below formula. Leaf count of a tree = Leaf count of left subtree + Leaf count of right subtree	O(n) O(n)	
4	Convert a Binary Tree into its Mirror Tree	Method 1: Recursive (1) Call Mirror for left-subtree i.e., Mirror(left-subtree) (2) Call Mirror for right-subtree i.e., Mirror(right-subtree) (3) Swap left and right subtrees. temp = left-subtree left-subtree = right-subtree right-subtree = temp Method 2: Iterative. Use queue and do a level order traversal. Before adding child to queue swap them and add child to queue	O(n)	
5	Find the node with minimum value in a Binary Search Tree	Do inorder traversal go to the left most node in tree, that is the smallest value in tree	O(n)	

6 14/01/20	Construct a special tree from given preorder traversal (https://www.geeksforgeeks.org/construct-a-special-tree-from-given-preorder-traversal/) Trick question	Method 1: Since a node can have no children or 2 children start from root (1st node in pre array) recursively add children only if the node is not a leaf node(leaf or not can be found in preLN array)	O(n)	
7	Convert Ternary Expression to Binary Tree	Code is wrong. https://www.geeksforgeeks.org/convert-ternary-expression-binary-tree/ Check and write your own code		
8	Diagonal Sum In Binary Tree	The idea is to keep track of vertical distance from top diagonal passing through root. We increment the vertical distance we go down to next diagonal. 1. Add root with vertical distance as 0 to the queue. 2. Process the sum of all right child and right of right child and so on. 3. Add left child current node into the queue for later processing. The vertical distance of left child is vertical distance of current node plus 1. 4. Keep doing 2nd, 3rd and 4th step till the queue is empty.	Not sure	

9	Foldable Binary Trees	Method 1:
		1) If tree is empty, then return true.
		2) Convert the left subtree to its mirror image
		mirror(root->left); /* See pblm 4 */
		3) Check if the structure of left subtree and right subtree is same
		and store the result.
		res = isStructSame(root->left, root->right); /*isStructSame()
		recursively compares structures of two subtrees and returns
		true if structures are same */
		4) Revert the changes made in step (2) to get the original tree.
		mirror(root->left);
		5) Return result res stored in step 2.
		
		Method 2:
		// Checks if tree can be folded or not
		
		IsFoldable(root)
			O(n)
			
			O(n)	

		1) If tree is empty then return true 2) Else check if left and right subtrees are structure wise mirrors of each other. Use utility function IsFoldableUtil(root->left, root->right) for this. // Checks if n1 and n2 are mirror of each other. IsFoldableUtil(n1, n2) 1) If both trees are empty then return true. 2) If one of them is empty and other is not then return false. 3) Return true if following conditions are met a) n1->left is mirror of n2->right b) n1->right is mirror of n2->left		
10	Extract Leaves of a Binary Tree in a Doubly Linked List (in place)	Do inorder traversal, maintain a previous pointer at each leaf node, once we move to next leaf assign current left to prev, prev right to current	O(n)	
11	Convert a Binary Tree to a Circular Doubly Link List	Can use modified inorder traversal, D:\preparation\Rent-a-desk\src\Tree.java	O(n)	
12	Postorder Traversal	Very basic using recursion	O(n)	
13	PreOrder Traversal	Very basic using recursion	O(n)	
14 15/01/20	Level with maximum number of nodes	Use level order traversal. Initially add root in queue and mark level as zero. Now size of queue gives the node of node at this level. Now using that node count to iterate remove the nodes from queue and add the child to queue. ONce node count is zero break the inner loop. Again do the same process as above. https://www.geeksforgeeks.org/level-maximum-number-nodes/	O(n)	

15	Evaluation of Expression Tree	If t is not null then
		  If t.info is operand then
		    Return t.info
		  Else
		    A = solve(t.left)
		    B = solve(t.right)
		    return A operator B
		    where operator is the info contained in t
			O(n)	

16	Sum of Binary Tree	The idea is to recursively, call left subtree sum, right subtree sum and add their values to current node's data.	O(n)	

17 16/01/20	Given a binary tree, how do you remove all the half nodes	Do postorder order traversal, If node is null return null If node left and right child are null return node. If node left child is null return right child If node right child is null return left If none of above matches return node.	O(n)	
18	Print nodes at k distance from root	Assume root at zero level, recursively at each level increment the count and check if level is k. if its k print data.	O(n)	
19	Vertical Sum in a given Binary Tree	We need to use horizontal distance (HD) to solve this plm. Root is at hd 0, left child is at –1(0-1) and right child is at +1(0+1). Similarly each left we substract 1 and on each righ we add 1. maintain a hash map with HD as key and sum as value. At the end print the map. (we can use inorder traversal) Other method doesn't use extra space for hashmap. Check later https://www.geeksforgeeks.org/vertical-sum-in-binary-tree-set-space-optimized/	O(n)	
20	Construct Complete Binary Tree from its Linked List Representation	1. Create an empty queue. 2. Make the first node of the list as root, and enqueue it to the queue. 3. Until we reach the end of the list, do the following. ………a. Dequeue one node from the queue. This is the current parent. ………b. Traverse two nodes in the list, add them as children of the current parent. ………c. Enqueue the two nodes into the queue.	O(n)	
21	Find maximum level sum in Binary Tree	Do a level order traversal, for each level calculate the sum and compare it with existing result.	O(n)	
22 17/	Maximum width of a binary tree	Same as above. Do a level order traversal, queue size gives us the no of nodes at each level, get max widht from queue. Method 2: recursive method where we pass the level to recursive function. We need a array with size equal to the height of tree. At each	O(n) O(n)	

0 1 / 2 0		node in tree increment the level count in arr. At the end find max element in array		
2 3	Vertical width of Binary tree	Take inorder traversal and then take a temporary variable if we go left then temp value decreases and if go to right then temp value increases. Assert a condition in this, if the minimum is greater than temp, then minimum = temp and if maximum less then temp then maximum = temp. In the end, print minimum + maximum which is the vertical width of the tree.	O(n)	
2 4 1 8 / 0 1 / 2 0	Postorder traversal from given Inorder and Preorder traversals	REVISIT		
2 5	Check whether a given binary tree is perfect or not	REVISIT Find depth of any node (in below tree we find depth of leftmost node). Let this depth be d. Now recursively traverse the tree and check for following two conditions. Every internal node should have both children non-empty All leaves are at depth 'd'		

2
6
	Difference between sums of odd level and even level nodes of a Binary Tree	Method 1:
		Do a level order traversal, at each level add the node data to odd sum or even sum. At the end return the difference.
		
		Method 2:
		The problem can also be solved using simple recursive traversal. We can recursively calculate the required difference as, value of root's data subtracted by the difference for subtree under left child and the difference for subtree under right child.
		int getLevelDiff(Node node)
		    {
		        // Base case
		        if (node == null)
			O(n)
			
			O(n)	

		return 0;
		
		// Difference for root is root's data - difference for
		// left subtree - difference for right subtree
		return node.data - getLevelDiff(node.left) -
		
		getLevelDiff(node.right);
		}		

27	Sum of all the numbers that are formed from root to leaf paths	int treePathsSumUtil(Node node, int val)
		{
		if (node == null)
		return 0;
		val = (val * 10 + node.data);
		
		if (node.left == null && node.right == null)
		return val;
		
		return treePathsSumUtil(node.left, val)
		+ treePathsSumUtil(node.right, val);
		}
			O(n)	

28	Check whether a binary tree is a full binary tree or not	1) If a binary tree node is NULL then it is a full binary tree.
		2) If a binary tree node does have empty left and right sub-trees, then it is a full binary tree by definition.
		3) If a binary tree node has left and right sub-trees, then it is a part of a full binary tree by definition. In this case recursively check if the left and right sub-trees are also binary trees themselves.
		4) In all other combinations of right and left sub-trees, the binary tree is not a full binary tree.
		
		CheckFBT(Node node)
		{
		If(node ==null) return true;
		If(node.left==null && node.right==null) return true;
		If(node.left!=null && node.right!=null)
		Return checkFBT(node.left) &&
		CheckFBT(node.right);
		Return false;
		}
			O(n)	

29	Check for Children Sum Property in a Binary Tree	Traverse the given binary tree. For each node check (recursively) if the node and both its	O(n)	

		children satisfy the Children Sum Property, if so then return true else return false.
		Public int isSumProperty(Node node)
		{
		    If(node == null) ||
		(node.left==null &&
		node.right==null) return 1;
		else
		    If(node.left!=null)
		      Left = node.left.data;
		    If(node.right!=null)
		      right = node. Right.data;
		
		    If((node.data = left+right)
		      &&
		isSumProperty(node.left)!=0
		      &&
		isSumProperty(node.right)!=0)
		      Return 1;
		   Else
		      Return 0;
		}		

30	Write Code to Determine if Two Trees are Identical	sameTree(tree1, tree2)
		1. If both trees are empty then return 1.
		2. Else If both trees are non -empty
		   (a) Check data of the root nodes (tree1->data == tree2->data)
		   (b) Check left subtrees recursively i.e., call sameTree(
		       tree1->left_subtree, tree2->left_subtree)
		   (c) Check right subtrees recursively i.e., call sameTree(
		       tree1->right_subtree, tree2->right_subtree)
		   (d) If a,b and c are true then return 1.
		3 Else return 0 (one is empty and other is not)
		
		boolean identicalTrees(Node a, Node b)
		  {
		     /*1. both empty */
		     if (a == null && b == null)
		       return true;
		
		     /* 2. both non-empty -> compare them */
		     if (a != null && b != null)
		       return (a.data == b.data
		           && identicalTrees(a.left, b.left)
		           && identicalTrees(a.right, b.right));
			O(n)	

		`/* 3. one empty, one not -> false */`
		`return false;`
		`}`		

31	Find Minimum Depth of a Binary Tree	Can be achieved In mutiple ways. 2 recursive methods and 1 level order traversal based. https://www.geeksforgeeks.org/find-minimum-depth-of-a-binary-tree/	O(n) For all 3 varients	
32	Right view of Binary Tree using Queue	Method 1: Simple solution is to do a level order traversal and print last node at each level. Method 2: The problem can also be solved using simple recursive traversal. We can keep track of level of a node by passing a parameter to all recursive calls. The idea is to keep track of maximum level also. And traverse the tree in a manner that right subtree is visited before left subtree. Whenever we see a node whose level is more than maximum level so far, we print the node because this is the last node in its level (Note that we traverse the right subtree before left subtree)	O(n)	

33	Symmetric Tree (Mirror Image of itself)	`boolean isMirror(Node node1, Node node2)`
		`{`
		`// if both trees are empty, then they are mirror image`
		`if (node1 == null && node2 == null)`
		`return true;`
		
		`// For two trees to be mirror images, the following three`
		`// conditions must be true`
		`// 1 - Their root node's key must be same`
		`// 2 - left subtree of left tree and right subtree`
		`//     of right tree have to be mirror images`
		`// 3 - right subtree of left tree and left subtree`
		`//     of right tree have to be mirror images`
		`if (node1 != null && node2 != null && node1.key == node2.key)`
		`return (isMirror(node1.left, node2.right)`
		`&& isMirror(node1.right, node2.left));`		

		// if neither of the above conditions is true then
		// root1 and root2 are mirror images
		return false;
		}		

34	Count BST nodes that lie in a given range	// Returns count of nodes in BST in
		// range [low, high]
		int getCount(Node node, int low, int high)
		{
		// Base Case
		if(node == null)
		return 0;
		
		// If current node is in range, then
		// include it in count and recur for
		// left and right children of it
		if(node.data >= low && node.data <= high)
		return 1 + this.getCount(node.left, low, high)+
		this.getCount(node.right, low, high);
		
		// If current node is smaller than low,
		// then recur for right child
		else if(node.data < low)
		return this.getCount(node.right, low, high);
		
		// Else recur for left child
		else
		return this.getCount(node.left, low, high);
		}		

35	Lowest Common Ancestor in a Binary Search Tree.	Traverse the tree and find a node where one is less than node data and other is greate than node data.	O(n)	

36	Convert a given tree to its Sum Tree	Do a traversal of the given tree. In the traversal, store the old value of the current node, recursively call for left and right subtrees and change the value of current node as sum of the values returned by the recursive calls. Finally return the sum of new value and value
		int toSumTree(Node node)
		{
		// Base case
		if (node == null)
		return 0;
		
		// Store the old value
			O(n)	

		int old_val = node.data;
		
		// Recursively call for left and right subtrees and store the sum
		// as new value of this node
		node.data = toSumTree(node.left) + toSumTree(node.right);
		
		// Return the sum of values of nodes in left and right subtrees
		// and old_value of this node
		return node.data + old_val;
		}		

3719/01/20	Tree Isomorphism Problem	Two trees are called isomorphic if one of them can be obtained from other by a series of flips, i.e. by swapping left and right children of a number of nodes. Any number of nodes at any level can have their children swapped. Two empty trees are isomorphic. Method 1: 1) Data of n1 and n2 is same. 2) One of the following two is true for children of n1 and n2 ……a) Left child of n1 is isomorphic to left child of n2 and right child of n1 is isomorphic to right child of n2. ……b) Left child of n1 is isomorphic to right child of n2 and right child of n1 is isomorphic to left child of n2.	O(n)	
38	Count Number of SubTrees having given Sum	REVISIT		
39	Get Level of a node in a Binary Tree	The idea is to start from the root and level as 1. If the key matches with root's data, return level. Else recursively call for left and right subtrees with level as level + 1. Method 2: use level order traversal, add null after each level to identify the end of levels. Use a variable level to track the current level.	O(n) O(n)	
4020/	How to determine if a binary tree is height-balanced?	**Very imp forms the basis for lots of other pblms** Method 1: get the height of left and right subtrees. Return true if difference between	O(n2)	Sol lin https://www.geeksforgeeks.org/how-

01/20		heights is not more than 1 and left and right subtrees are balanced, otherwise return false.
		
		Method 2: In above method we calculate the height of left and right subtrees at each level. We can use the same recursion to calculate height too
		
		Public boolean isBalanced(node, height)
		{
		    If(node == null)
		    {
		        Height = 0; return true;
		    }
		    Height lheight = new height();
		    Height rheight = new height();
		    Boolean l = isBalanced(node, lheight);
		    Boolean r = isBalanced(node, rheight);
		    Int lh = lheight.height;
		    Int rh = rheight.height;
		    Height = (lh>rh?lh:rh)+1;
		     If(Math.abs(lh-rh)>=2) return false;
		    Return l && r;
		}
			O(n)	to-determine-if-a-binary-tree-is-balanced/

41	Diameter of the Tree	REVISIT		
42	Binary Tree to Binary Search Tree Conversion	1) Create a temp array arr[] that stores inorder traversal of the tree. This step takes O(n) time. 2) Sort the temp array arr[]. Time complexity of this step depends upon the sorting algorithm. In the following implementation, Quick Sort is used which takes (n^2) time. This can be done in O(nLogn) time using Heap Sort or Merge Sort. 3) Again do inorder traversal of tree and copy array elements to tree nodes one by one. This step takes O(n) time.		
43	Check if two trees are Mirror	Method 1: Use recursive approach and check the following If(node1.data==node2.data && isMirror(node1.left, node2.right) && isMirror(node1.right, node2.left)) Method 2: Do Inorder traversal of tree1 and reverse Inorder traversal of 2nd tree and compare both the elements, if both of elements are equal then proceed further	O(n) O(n)	

44	Merge two BSTs with limited extra space	We need to follow the merge method logic. Using 2 stacks do iterative inorder travesal and print the elements		
45	Find the maximum path sum between two leaves of a binary tree	**REVISIT**		
46 09/03/20	Find the closest element in Binary Search Tree	Method 1: A simple solution for this problem is to store Inorder traversal of given binary search tree in an auxiliary array and then by taking absolute difference of each element find the node having minimum absolute difference with given target value K in linear time. Method 2: If target value K is present in given BST, then it's the node having minimum absolute difference. If target value K is less than the value of current node then move to the left child. If target value K is greater than the value of current node then move to the right child.	O(h)	
47	Print BST keys in the given range	**Method 1:** The idea is to use the inorder traversal, as inorder traversal of a BST gives the keys in sorted order. 1) If value of root's key is greater than k1, then recursively call in left subtree. 2) If value of root's key is in range, then print the root's key. 3) If value of root's key is smaller than k2, then recursively call in right subtree.	O(n)	
48	Print extreme nodes of each level of Binary Tree in alternate order	**Method 1:** The idea is to traverse tree level by level. For each level, we count number of nodes in it and print its leftmost or the rightmost node based on value of a Boolean flag. We dequeue all nodes of current level and enqueue all nodes of next level and invert value of Boolean flag when switching levels.	O(n)	

49	Find maximum (or minimum) in Binary Tree	In Binary Search Tree, we can find maximum by traversing right pointers until we reach the rightmost node. But in Binary Tree, we must visit every node to figure out maximum. So the idea is to traverse the given tree and for every node return maximum of 3 values. 1) Node's data. 2) Maximum in node's left subtree. 3) Maximum in node's right subtree.		
50	Preorder to Postorder	**REVISIT**		
51	Sum of leaf nodes at minimum level	Perform iterative level order traversal using queue and find the first level containing a leaf node. Sum up all the leaf nodes at this level and then stop performing the traversal further.	Time Complexity: O(n). Auxiliary Space: O(n).	
52	Print leftmost and rightmost nodes of a Binary Tree	The idea is to use Level Order Traversal. Every time we store the size of the queue in a variable n, which is the number of nodes at that level. For every level we check three conditions, whether there is one node or more than one node, in case there is only one node we print it once and in case we have more than 1 nodes, we print the first (i.e node at index 0) and the node at last index (i.e node at index n-1).	O(n)	