	Problem	Solution	Time Complexity	Failure case
1	Given an array of size N which represents the elements to be inserted into BST (considering first element as root). The task is to find the minimum element in this given BST. If the tree is empty, there is no minimum element, so print -1 in that case	Build a BST from given array. Find the left most element and return.	Worst case O(n) for skewed tree	
2	Construct a Balanced BST from sorted array	1) Get the Middle of the array and make it root. 2) Recursively do same for left half and right half. a) Get the middle of left half and make it left child of the root created in step 1. b) Get the middle of right half and make it right child of the root created in step 1.	O(n)	
3	Inorder traversal of BST	InOrder Traversal of BST always returns a sorted list		
4.	Convert a BST to Balanced BST (Skewed BST to a balanced BST)	Method 1: Do a inorder traversal of tree and add each element into a balanced tree like AVL tree. Method 2: Do a inorder traversal and generate a sorted array. This takes O(n) time. Now construct Balanced BST from sorted array (see pblm 2) Time complexity O(n)	O(nlogn) not guaranteed O(n)	
5.	Search a node in BST	To search a given key in Binary Search Tree, we first compare it with root, if the key is present at root, we return root. If key is greater than root's key, we recur for right subtree of root node. Otherwise we recur for left subtree.	The worst case time complexity of search and insert operations is O(h) where h is height of BST In worst case, we may have to travel from root to the deepest leaf node. The	

			height of a skewed tree may become n and the time complexity of search and insert operation may become O(n).	
6.	Given an integer N, how many structurally unique binary search trees are there that store values 1...N?	REVISIT		
7.	Lowest Common Ancestor in a Binary Search Tree.	Start from root. Compart n1 and n2 with root. If n1 or n2 is equal to root then return root. If n1<root <n2 then root is LCA If n1 and n2 < root search left subtree for LCA If n1 and n2 > root search right subtree for LCA	O(h) where h is height of tree	
8.	Add all greater values to every node in a BST	do reverse Inorder traversal of BST, we get all nodes in decreasing order. We do reverse Inorder traversal and keep track of the sum of all nodes visited so far, we add this sum to every node.	O(n)	
9.	Binary Tree to BST	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. O(nLogn) time for 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.		
10 19 /	Merge Two Balanced Binary Search Trees	Method 1: Insert the elements of one tree into other tree Method 2:	mLog(m+n-1).	

12/19		1) Do inorder traversal of first tree and store the traversal in one temp array arr1[]. This step takes O(m) time. 2) Do inorder traversal of second tree and store the traversal in another temp array arr2[]. This step takes O(n) time. 3) The arrays created in step 1 and 2 are sorted arrays. Merge the two sorted arrays into one array of size m + n. This step takes O(m+n) time. 4) Construct a balanced tree from the merged array using the technique discussed pblm 2. This step takes O(m+n) time. Method 3 (In-Place Merge using DLL) We can use a Doubly Linked List to merge trees in place. Following are the steps. 1) Convert the given two Binary Search Trees into doubly linked list in place 2) Merge the two sorted Linked Lists 3) Build a Balanced Binary Search Tree from the merged list created in step 2.	O(m+n) O(m+n)	
11	Merge two BST 's	REVISIT. V.imp google,microsoft , amazon question.		
12	Two nodes of a BST are swapped, correct the BST	Method 1: Do inorder traversal and store in a array. Sort the array and do inorder travesal and replace array elements in tree. Method 2: maintain three pointers, first, middle and last. When we find the first point where current node value is smaller than	O(nlogn) O(n)	

		previous node value, we update the first with the previous node & middle with the current node. When we find the second point where current node value is smaller than previous node value, we update the last with the current node. If adjacent nodes are swapped, we will never find the second point. So, last pointer will not be updated. After processing, if the last node value is null, then two swapped nodes of BST are adjacent.		
13	Find the Closest Element in BST	Method 1: 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. At each step do node.data-k and compare it with min value. If less than min store other wise ignore.	O(h) Where h is the height if the tree	
14	Print BST keys in the given range	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.	O(n)	

		3) If value of root's key is smaller than k2, then recursively call in right subtree.		
15	Find k-th smallest element in BST	Method 1: Do inorder traversal, keep count of nodes visited, when count reaches k pint it. Method 2: Augmented Tree DS. This is spl kind of BST where we maintain the count of nodes in left subtree for each node. Assume that the root is having N nodes in its left subtree. If K = N + 1, root is K-th node. If K < N, we will continue our search (recursion) for the Kth smallest element in the left subtree of root. If K > N + 1, we continue our search in the right subtree for the (K – N – 1)-th smallest element. Note that we need the count of elements in left subtree only.	O(n) O(h)	
16	Find median of BST in O(n) time and O(1) space	REVISIT https://www.geeksforgeeks.org/find-median-bst-time-o1-space/		
17	Preorder to Postorder	REVISIT		
18	Construct BST from Postorder	REVISIT		
19	Level Order Traversal	Method 1: O(n2) solution uses 2 funstions PrintGivenLevel and printLevelOrder. Method 2: Using Queue 1) Create an empty queue q 2) temp_node = root /*start from root*/ 3) Loop while temp_node is not NULL a) print temp_node->data.		

		b) Enqueue temp_node's children (first left then right children) to q c) Dequeue a node from q and assign it's value to temp_node		
2 0	Check if a BST is complete BST	The approach is to do a level order traversal starting from the root. In the traversal, once a node is found which is NOT a Full Node, all the following nodes must be leaf nodes. Also, one more thing needs to be checked to handle the below case: If a node has an empty left child, then the right child must be empty. Method 2: A recursive solution exists check if possible	O(n)	
2 1	Check if a given Binary Tree is Heap	We need to divide this problem in to 2 parts. IsCompleteBST and isBSTHeap. IsCompleteBST see pblm 20. IsBSTHeap: Starting from non leaf nodes (leaves start at n/2+1) see if node.data is greater than left and right child data	O(n)	
2 2 1 1 / 1 / 2 0	Find a pair with given sum in a Balanced BST Expected : time complexity : O(n) Space complexity : O(Logn) Modification to Binary Search Tree is not allowed.	Method 1: Brute force method For each node in tree, traverse to see if a value exists where sum of values is zero Method 2: Do a inorder traversal and store values in array. (inorder traversal gives a sorted array). Next traverse array from both sides and check if a pair with zero sum exists. Method 3: Convert a BST to DLL inplace. Next traverse DLL from both sides and check if a pair with givem sum exists.	Time complexity O(n2) Time complexity O(n) Space complexity O(n) Time complexity O(n) Doesn't use any extra space. But this modifies the BST	

		Method 4: Run inorder and reverse inorder traversal simultaneously. At each step see if sum of 2 node values is equal to given sum. If its greater only do reverse inorder, if its less do a inorder .	Time Complexity O(n) No extra space.	
23	You are given a Binary tree. You are required to find the number of pairs violating the BST property.	Store the in-order traversal of the binary tree in an array. Now, count the number of inversions (i.e. a[i] > a[j] and i < j) in this array using Mergesort algorithm.	Time Complexity O(nlogn).	
24	Given a BST and a value x, the task is to delete the nodes having values greater than or equal to x.	do a post order traversal of the tree and delete all the nodes which are greater than or equal to the value of k.	Time Complexity O(nlogn).	
25	Inorder predecessor and successor for a given key in BST	1. If root is NULL then return 2. if key is found then a. If its left subtree is not null Then predecessor will be the right most child of left subtree or left child itself. b. If its right subtree is not null The successor will be the left most child of right subtree or right child itself. return 3. If key is smaller then root node set the successor as root search recursively into left subtree else set the predecessor as root search recursively into right subtree		

26	K'th Largest Element in BST when modification to BST is not allowed	The idea is to do reverse inorder traversal of BST. The reverse inorder traversal traverses all nodes in decreasing order. While doing the traversal, we keep track of count of nodes visited so far. When the count becomes equal to k, we stop the traversal and print the key.	O(h + k)	
27	Given an array arr of size n, write a program that prints 1 if given array can represent preorder traversal of a BST, else prints 0.	REVISIT		
28	How to check if a given array represents a Binary Heap?	Method 1: Start from 1st element check if all the elements to its right are less than current element. Method 2: Instead of checking all the elements, check only the children of each node. For any node at position I left child is at 2*i+1 and right is 2*I+2 position	O(n2) O(n)	
29 13/01/20	Inorder Successor in Binary Search Tree	Method 1: This method assumes there is a parent pointer at each node which points to the parent of each node. 1. If right subtree is not null then go to right subtree and return the smallest element in right subtree. (traverse left util its not null. Last non null element is smallest) 2. If right sub tree is null then, succ lies above the node. traverse using the parent pointer until the node is left child of the root. The parent of such a node is the succ.	O(h) where h is height of tree	

		Method 2: This method doesn't need a parent pointer. 1. If right subtree is not null then go to right subtree and return the smallest element in right subtree. (traverse left util its not null. Last non null element is smallest) 2. If right sbtree of node is NULL, then start from root and us search like technique. Do following. Travel down the tree, if a node's data is greater than root's data then go right side, otherwise go to left side.	O(h) where h is height of tree	
30	Iterative Inorder traversal	1) Create an empty stack S. 2) Initialize current node as root 3) Push the current node to S and set current = current->left until current is NULL 4) If current is NULL and stack is not empty then a) Pop the top item from stack. b) Print the popped item, set current = popped_item->right c) Go to step 3. 5) If current is NULL and stack is empty then we are done.	O(n)	
31	Print Common Nodes in Two Binary Search Trees	REVISIT (Easy)		
32	AVL Tree insertion	REVISIT(Complex)		
33	Delete a Node from BST	1. I f node to be deleted is a leaf node, delete it. 2. If node to be deleted is non leaf and has only one child, delete node	O(h) where h is the height of the BST	

		and replace it with its child. 3. If node to be deleted has both left and eight children, find the inorder successor of the node. replace node with it inorder successor and delete the inoder successor node. (we can also use inorder predecesor in case right sub tree is null)		
34	Leaf nodes from Preorder of a Binary Search Tree	Method 1: How to traverse in preorder fashion using two arrays representing inorder and preorder traversals? We iterate the preorder array and for each element find that element in the inorder array. For searching, we can use binary search, since inorder traversal of binary search tree is always sorted. Now, for each element of preorder array, in binary search we set the range [L, R]. And when L == R, leaf node is found. So, initially, L = 0 and R = n – 1 for first element (i.e root) of preorder array. Now, to search for element on the left subtree of root, set L = 0 and R = index of root – 1. Also, for all element of right subtree set L = index of root + 1 and R = n -1. Recursively, follow this, until L == R. Method 2: There is other method, that doesn't seem to work for specific cases. Jst check once https://www.geeksforgeeks.org/leaf-nodes-preorder-binary-search-tree/	Time Complexity: O(n log n) Auxiliary Space: O(n)	

35	Array to BST	1) Get the Middle of the array and make it root. 2) Recursively do same for left half and right half. a) Get the middle of left half and make it left child of the root created in step 1. b) Get the middle of right half and make it right child of the root created in step 1.	O(n)	
36	Largest BST	REVISIT (complex)		
37	Find median of BST in O(n) time and O(1) space	Requires Morries Inorder traversal knowledge REVISIT		
38	Sum of k smallest elements in BST	The idea is to traverse BST in inorder traversal. Note that Inorder traversal of BST accesses elements in sorted (or increasing) order. While traversing, we keep track of count of visited Nodes and keep adding Nodes until the count becomes k. Other method exists which modifies the tree, check its time complexity is O(h)	O(k)	
39	Check for BST	Method 1: The trick is to write a utility helper function isBSTUtil(struct node* node, int min, int max) that traverses down the tree keeping track of the narrowing min and max allowed values as it goes, looking at each node only once. The initial values for min and max should be INT_MIN and INT_MAX — they narrow from there. Method 2: 1) Do In-Order Traversal of the given tree and store the result in a temp array.	O(n) O(n)	

		3) Check if the temp array is sorted in ascending order, if it is, then the tree is BST. We can avoid the use of Auxiliary Array. While doing In-Order traversal, we can keep track of previously visited node. If the value of the currently visited node is less than the previous value, then tree is not BST		