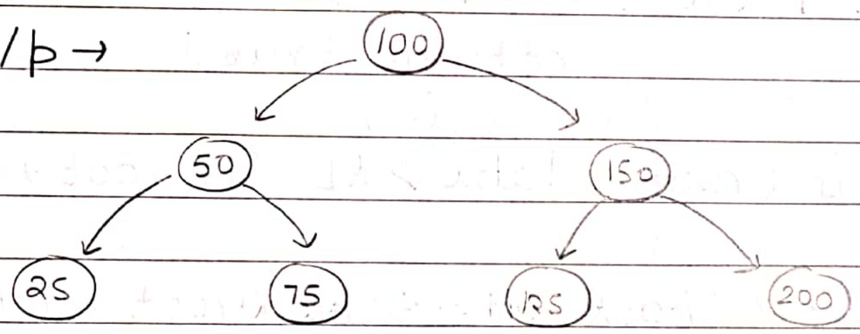
17/05/2023

Height and diameter of BST will be same as that of Binary trees.

Q1 Validate BST

i/p →



Approach-1 ⇒ Find inorder traversal & save it in the array and if the array is sorted, then it is a binary search tree. TC ⇒ O(n)

Approach-2 ⇒ Can we do in a single pass by recursion.

Root node (100) ⇒ $(-\infty, \infty)$ -2^{32} 2^{32}

50 ⇒ $(-\infty, 100)$

25 ⇒ $(-\infty, 50)$

75 ⇒ $(50, 100)$

150 ⇒ $(100, \infty)$

125 ⇒ $(100, 150)$

$200 \ni (150, \infty)$

All the values of nodes are in range & hence is a valid BST.

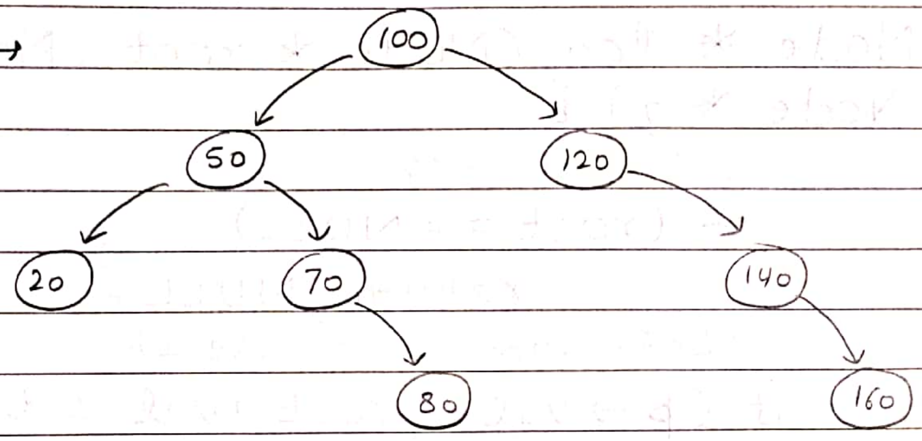
On moving left, upper bound changes.
On moving right, lower bound changes.

Code

```
bool solve (Node * root, long long int lb,
long long int ub) {
        // Empty tree is a BST
        if (root == NULL)
                return true;
        //Root has no limit
        if (root -> data > lb && root -> data < ub)
                //Check left subtree is BST or not
                bool lA = solve (root -> left, lb, root -> val
            //Check right subtree is BST or not
            bool rA = solve (root -> right, root -> val, ub)
            //Both right & left subtree should be BST
            return lA && rA;
    }
    else { //Not a BST
        return false;
    }
}
bool isValid BST (Node * root) {
    long long int lb = - 4294967296; } -2^32
    long long int ub = 4294967296; } 2^32
    bool ans = solve (root, lb, ub);
    return ans;
}
```

Q2 Lowest common ancestor of BST

i/p →

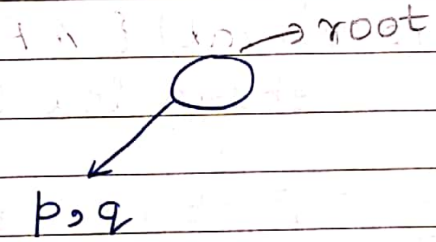


p = 70 , q = 160

We can find LCA via same method that we have done in binary trees but here we need to do something different as BST is given.

Ans → 100

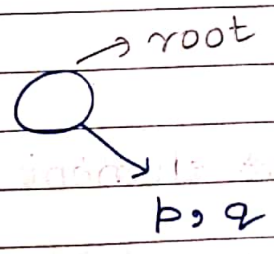
Case-1



p < root → data and q < root → data

Here left call will go.

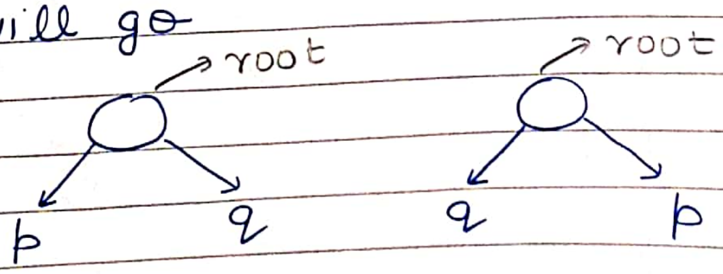
Case-2



p > root → data && q > root → data

Here right call will go

Case-3



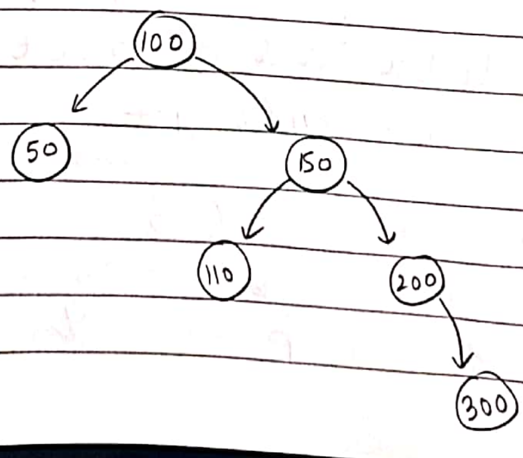
Ans = root

Code

```
Node * lca (Node * root, Node * p,
Node * q) {
    //Empty tree
    if (root == NULL)
        return NULL;
    //Left subtree (Case 1)
    if (p->val < root->val && q->val <
        root->val) {
        //Left call
        return lca (root->left, p, q);
    }
    // Right subtree (Case 2)
    else if (p->val > root->val && q->val >
        root->val) { // Right call
            return lca (root->right, p, q);
    }
    // Case 3 : Root is the answer
    else {
        return root;
    }
}
```

Q3 kth smallest element in BST

i/p →



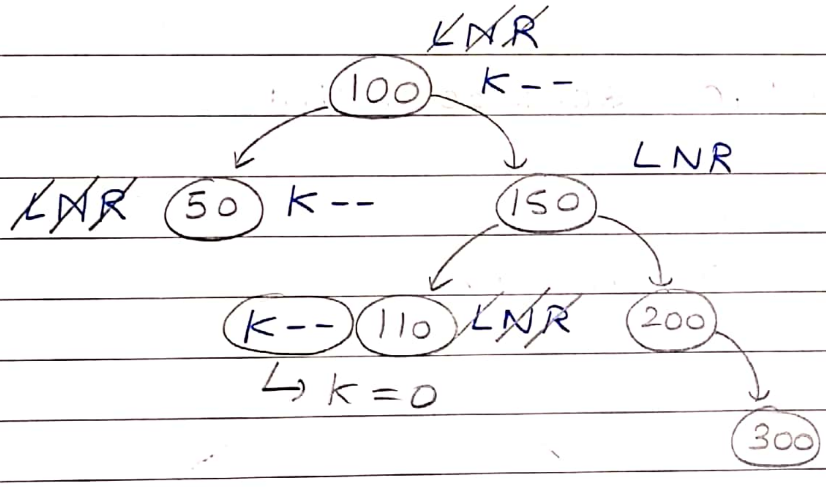
k = 3

Approach-1 ⇒ Store inorder traversal

50 100 110 150 200 300

↑ k = 3 (Ans)

Approach-2 ⇒ Do without storing the inorder traversal.



Hence 110 is the answer. When we have processed the node, then decrement k.

Code

```
int kthSmallest (Node * root, int &k) {
        // Base case
        if (root == NULL)
                return -1;
      // Left call
      int leftAns = kthSmallest (root->left, k);
      // Valid ans from left
      if (leftAns != -1)
              return leftAns;
    //Node
    k--;
    if (k == 0)
          return root->data;
```

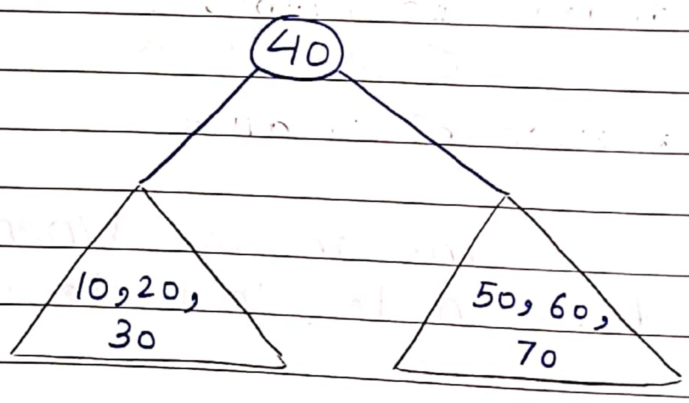
```
//Right call
int rightAns = kthSmallest (root->right, k);
return rightAns ;
}
```

Q4 Create a BST from inorder traversal.

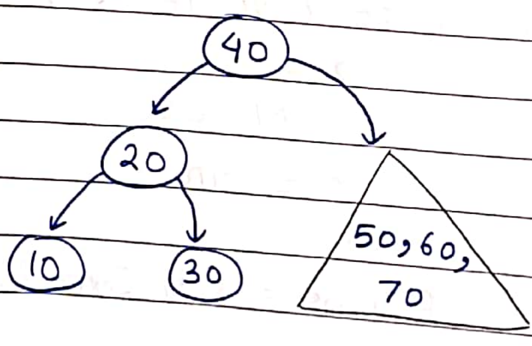
	0	1	2	3	4	5	6	7
i/p →	10	20	30	40	50	60	70	80

↳ Take it as root node

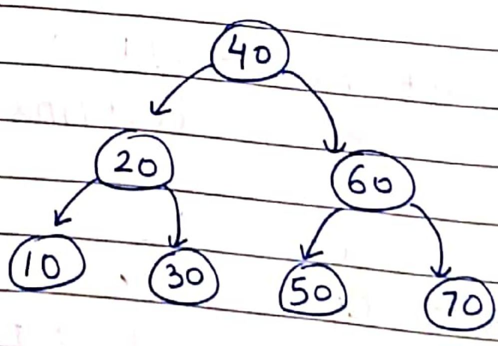
$mid = \frac{0+7}{2} = 3$



10, 20, 30
↳ mid



50, 60, 70
↳ mid



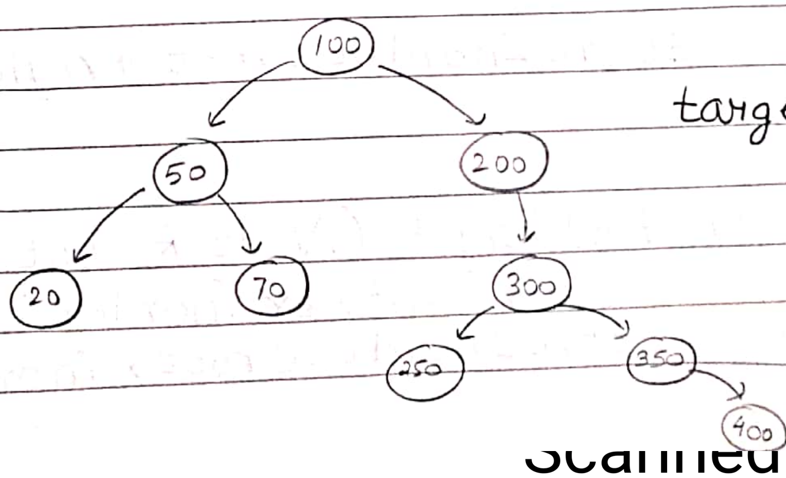
Code

```
Node * bstUsingInorder (int inorder [], int
s, int e) {
        // Base case
        if (s > e) //invalid array
            return NULL;
        // Find middle node
        int mid = s + (e-s)/2 ;
        int element = inorder [mid];
        // Create root element
        Node * root = new Node (element);
        //Left subtree creation
        root→left = bstUsingInorder (inorder, s, mid-1);
        //Right subtree creation
        root→right = bstUsingInorder (inorder, mid+1, e);
        return root;
}
```

Q5 Convert a BST into balanced BST.
Try the above question with the help of Q4 as we have made a balanced BST in that.

Q6 Two sum

i/p →



target = 320

We have to find 2 values which sum up to the target value.

Approach - 1

* 100, 320-100 = 220] search in BST
* 50, 320 - 50 = 270] search in BST
* 20, 320 - 20 = 300
↳ found in BST

Time complexity in average case = O(nlogn)

Approach-2

Store inorder traversal and then using 2 pointers find whether there exists 2 values which sum upto target or not.

Code

```
void storeInorder (Node * root, vector
<int> & inorder) {
        // Base case
        if (root == NULL)
                return;
        // Left
        storeInorder (root -> left, inorder);
        // Node
        inorder.push_back (root -> data);
        // Right
        storeInorder (root -> right, inorder);
}

bool findTarget (Node * root, int k) {
        vector <int> inorder;
        storeInorder(root, inorder);
```

225

```
//2 pointer approach
int s =0;
int e = inorder.size() -1;
while (s<e) {
    int sum = inorder [S] + inorder [e];
    if (sum == k) { //Found sum
        return true;
    }
    else if (sum > k) { // Smaller value needed
        e--;
    }
    else {    //Larger value needed
        s++;
    }
}
return false;
}
```