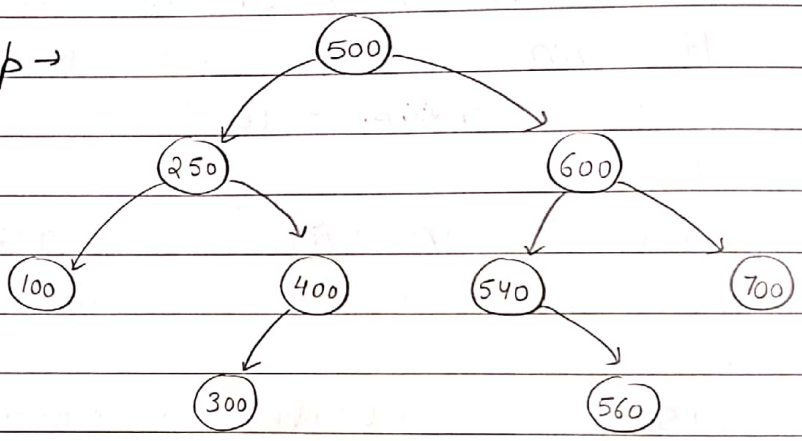
19/05/2023

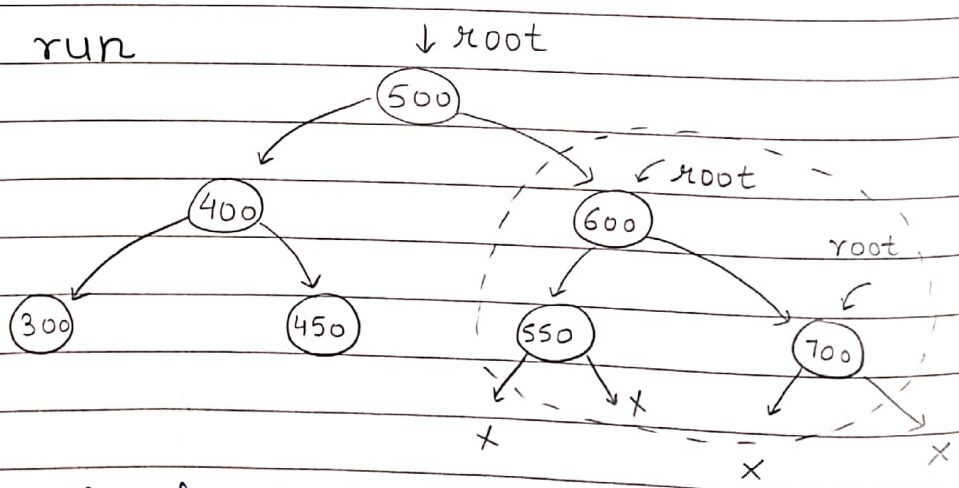
Q1 Convert BST into sorted doubly linked list

i/p →

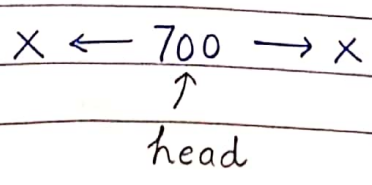


The question should be done in O(1) space. We have to think in such a way that left pointer is prev and right pointer is next. Think of solving one case and then recursion will handle.

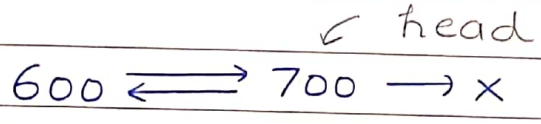
Dry run



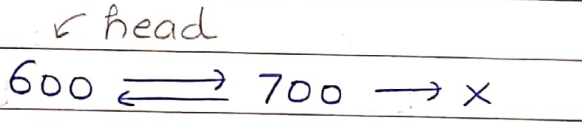
Initially head is NULL. Now we have 700 in the linked list having next and prev pointer as NULL.



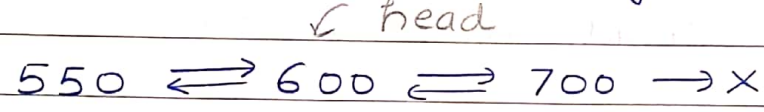
Now we have 600 in the linked list added. 600's right pointer is pointing to 700 i.e next pointer.



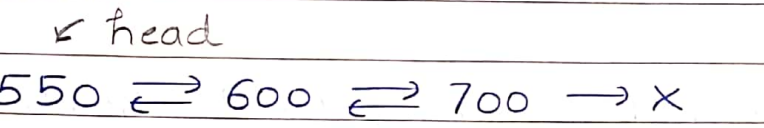
Update head to 600.



Now we have come to 550. It's right pointer is NULL & we checked whether head is NULL or not. head is not NULL & hence simply point this to head & 600 is pointing to 550



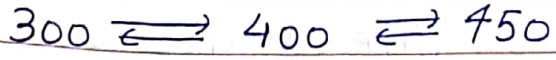
Update head



Now left pointer of 550 is NULL & there is nothing & hence right subtree has been converted to the Doubly linked list.

Note → In code we simply have to send recursive call for left subtree & right subtree which is converted to DLL. Now simply we have to put root (500) in between the 2 doubly linked list.

From left subtree we got

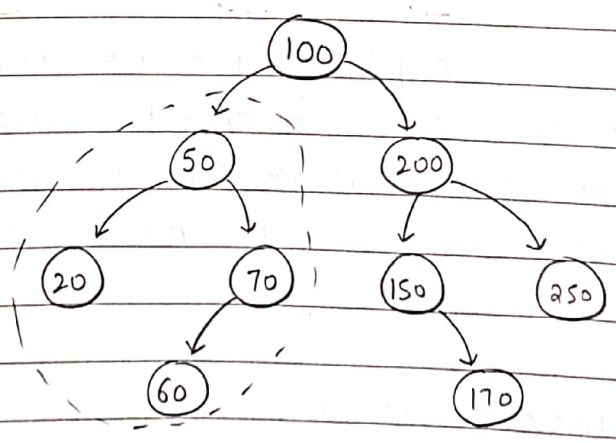


Simply put 500 in between the DLL or we have attached 500 already.

Code

```
void convertToDLL (Node * root, Node *& head)
    // Base case
    if (root == NULL)
            return;
    // Right subtree is getting converted to DLL
    convertToDLL (root→right, head);
    // Attach root node to right DLL
    root→right = head;
    // Attaching previous pointer
    if (head != NULL)
            head→left = root;
    // Update head
    head = root;
    // Convert left subtree to DLL.
    convertToDLL (root→left, head);
}
```

What if we solved left subtree first?



Suppose we got our linked list as

head ↙

$X \leftarrow 20 \rightleftarrows 50 \rightleftarrows 60 \rightleftarrows 70 \rightarrow X$

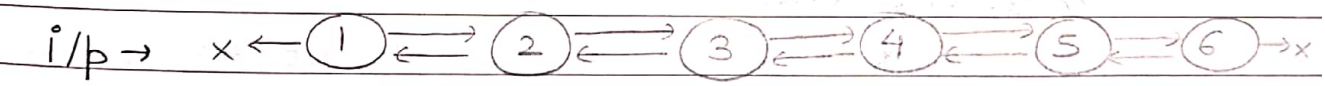
Now we have to attach 100 to 70. For that either we have to maintain the tail pointer or we have to traverse which is getting complex & hence it is better to solve right subtree first.

Steps

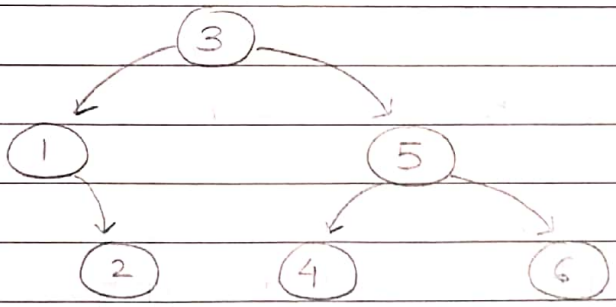
1) Solve right subtree
2) Attach root. (Both pointers)
3) Update head.
4) Solve left subtree.

TC = O(n)
SC = O(h)
↳ height (avg)
= O(n)
↳ Skew tree

Q2 Converted sorted linked list into BST.

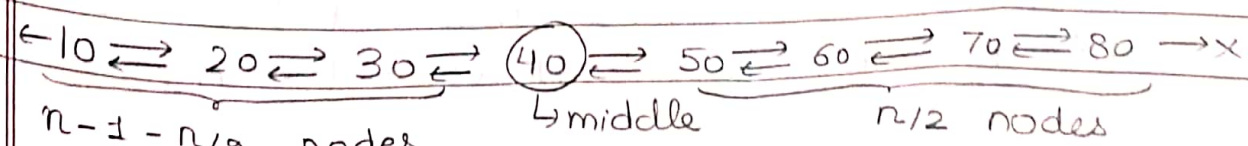


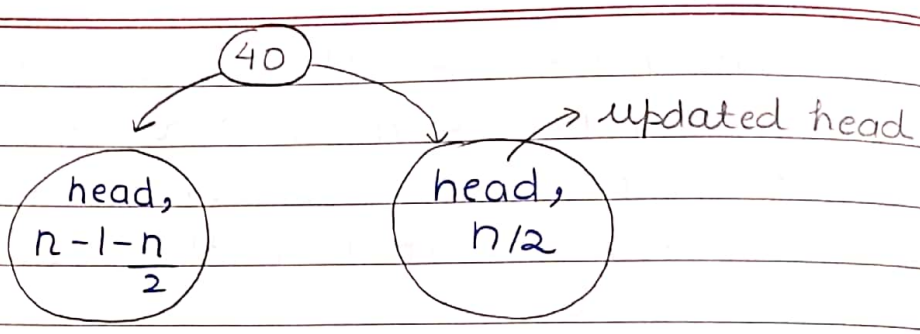
o/p →



We will be solving only one case i.e just make node for mid in LL and then we need to attach to the left & right subtrees which we get from recursion. Finding middle in a linked list can be done via slow & fast pointer.

Dry run





Code

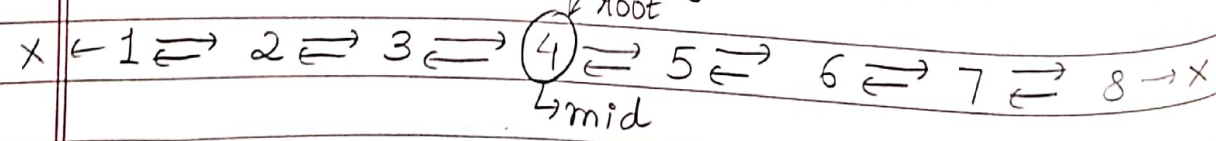
```
Node * sortedLLInto BST(Node * & head,
int n) {
  // Base case
  if (head == NULL  || n <=0)
                return NULL;
  // Getting left subtree
  Node * leftS=sortedLLIntoBST(head,
                              n-1-n/2);
  //Make root node as we are now at mid.
  Node * root = head;
  //Attach left subtree to root
  root → left = leftS;
  head = head → right; //head now pointing to
                       next of mid
  //Getting right subtree
  root → right = sortedLLIntoBST(head,n/2
  return root;
}
```

Mistake can happen here

$TC = O(n)$

$SC = O(h) = O(\log n)$

Understanding flow of code

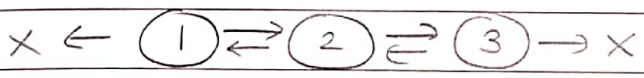


Here we are solving the left part as this is linked list & if we solve right

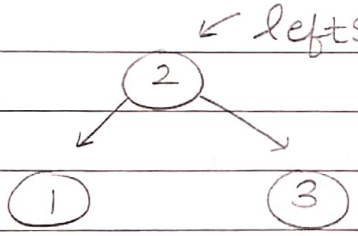
part first, this means we need to traversal & hence solving left part is beneficial.

In left subtree

$$n-1-\frac{n}{2} = 8-1-\frac{8}{2} = 7-4 = 3$$

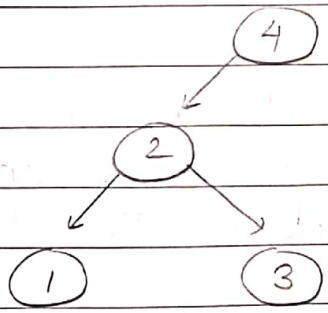


The above part of LL gets converted by recursion to

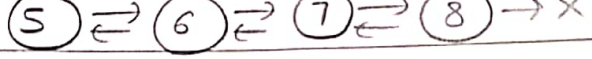


Now head will be pointing to mid & hence root is pointed to head.

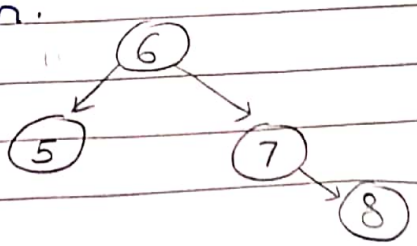
root → left = leftS;



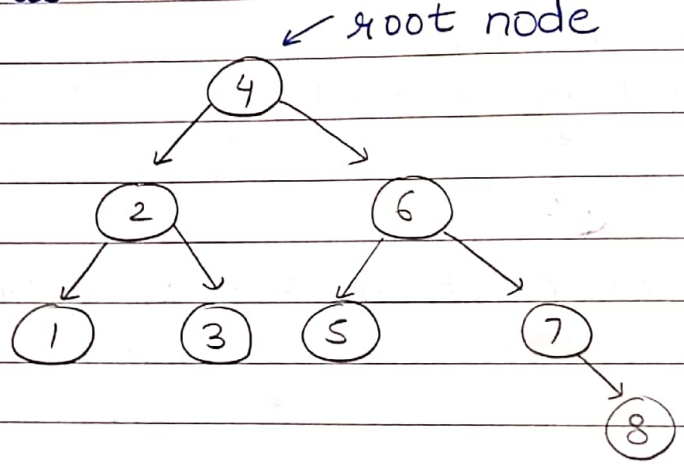
Now head updated to head → right. Now head is pointing to 5.



Now convert the above LL to tree & this is done by recursion.



Now root → right is already connected as we can see in LL. Hence we finally get the tree as



Note → We can update the recursive call as

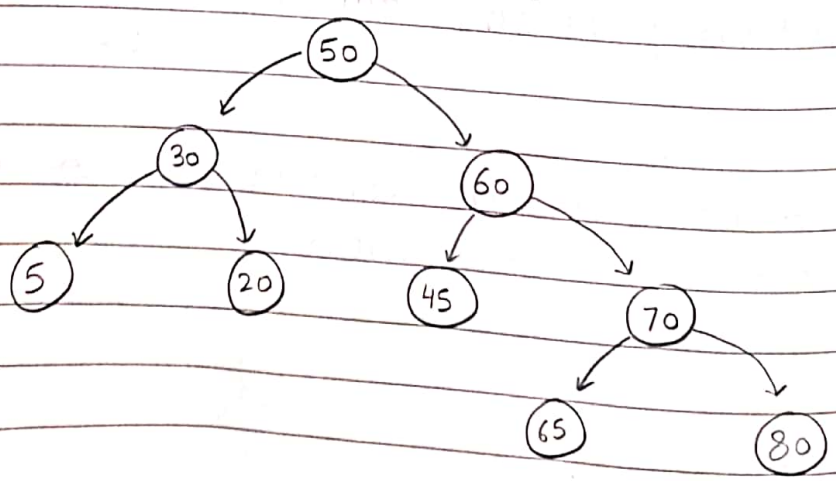
```
Node * leftS = SortedLLIntoBST(head, n/2);

root → right = SortedLLIntoBST(head, -n/2-1+n);
                                    n-1-n/2
```

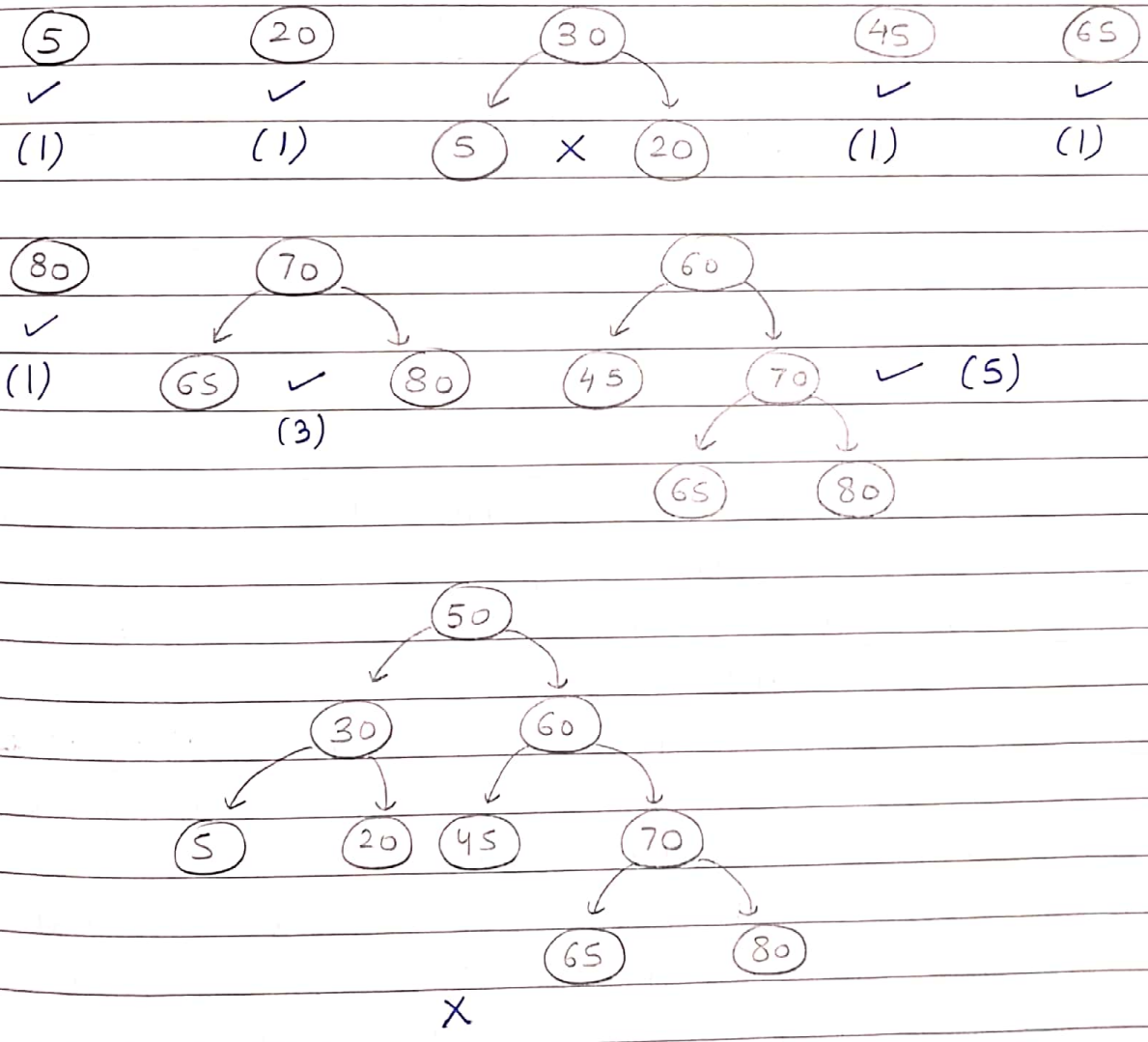
The above will create balanced BST for odd no. of nodes as well. Both the codes will create a valid BST.

Q3 Largest BST in a binary tree.

i/p →



The pattern of the question is basically same as that of fast method to find the diameter of the tree.
Binary tree is basically having only 0 / 1 or 2 child. In a given binary tree, find largest BST.

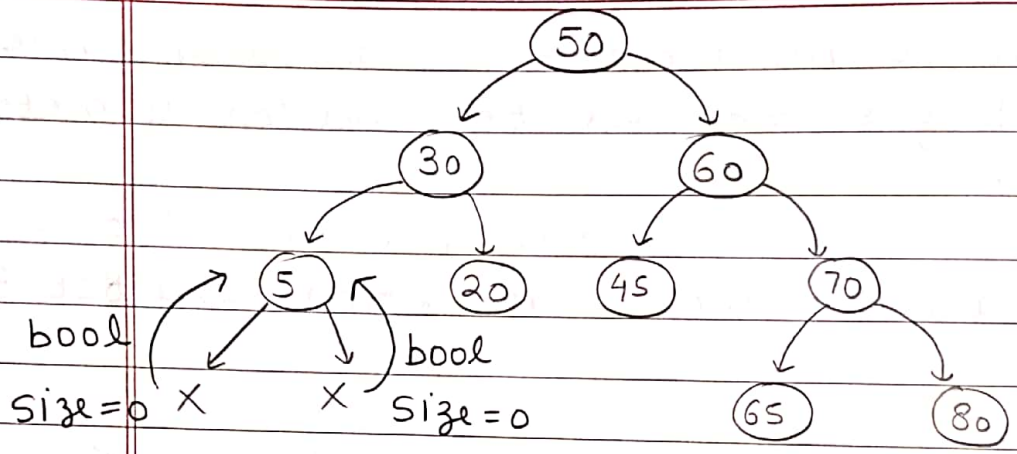


Hence no of nodes in largest BST is 5.

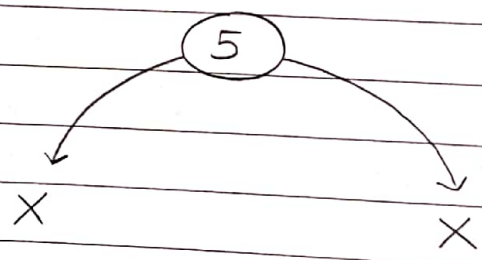
Approach-1 → Find inorder of the binary tree & find the largest subset which is sorted. (Check will it work or not)

Approach-2 → Let's follow bottom up approach here.

LRN



At node 5



1) Left subtree of 5 should be a BST and we get to know by some boolean variable.
2) Right subtree of 5 should be a BST.
3) In left subtree of 5, find maximum value and compare it with 5 & if it is smaller than 5, then satisfied. Here return maximum value.
4) In right subtree of 5, find smallest value and compare it with 5, if it is bigger than 5, then satisfied.
5) A node will also return sum of size i.e
0 + 0 + 1 = 1
↑ ↑ ↑ size of 5
null null

The above steps we need to follow for each & every node.

Code

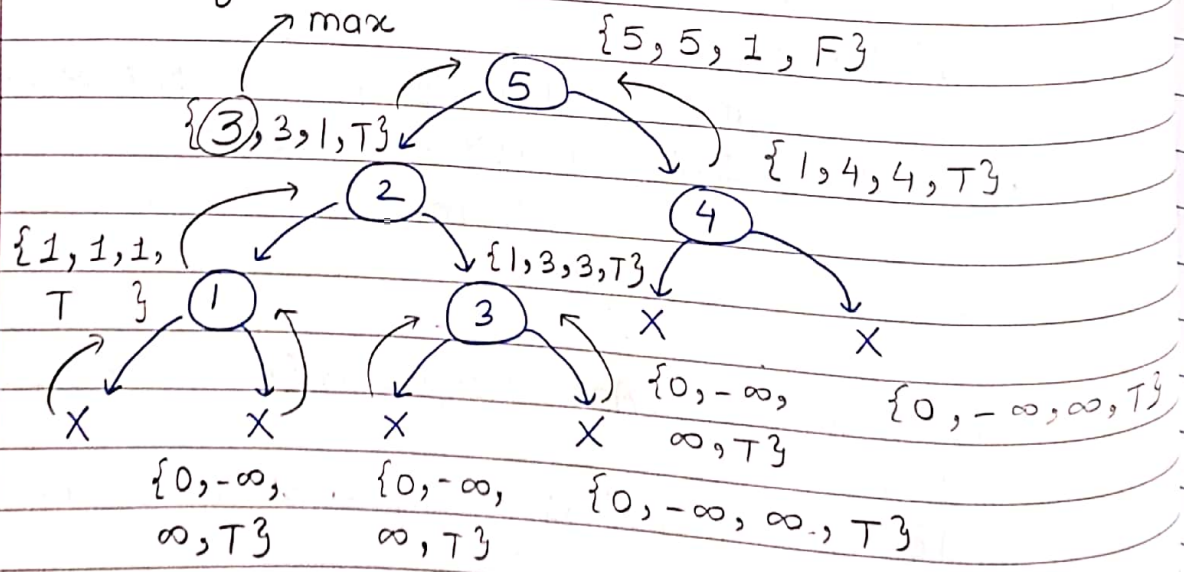
```
class NodeData {
    public:
        int size;
        int minVal;
        int maxVal;
        bool validBST;
        NodeData() {    }
        NodeData(int size, int min, int max,
        bool valid) {
            this->size = size;
            minVal = min;
            maxVal = max;
            validBST = valid;
        }
};

NodeData largestBST(Node *root, int &ans)
{
    // Base case
    if(root == NULL) {
        NodeData temp(0, INT_MIN,
        INT_MAX, true);
        return temp;
    }
    // Left subtree
    NodeData leftAns = largestBST(root->left, ans);
    // Right Subtree
    NodeData rightAns = largestBST(root->right, ans);
    // Checking starts
    NodeData currNode;
    currNode.size = 1 + leftAns.size + rightAns.size;
```

```
currNode.maxVal = max(root->data, rightAns.
 maxVal);
currNode.minVal = min(root->data, leftAns.
 minVal);
//Left, Right & current node should be BST
if(leftAns.validBST && rightAns.validBST
 && (root->data > leftAns.maxVal && root->
 data < rightAns.minVal)){
    //Valid BST
    currNode.validBST = true;
}
else { //Not a valid BST
    currNode.validBST = false;
}
if (currNode.validBST) //Largest BST to
    ans = max(ans, currNode.size); take
return currNode;
}
```

TC = O(n)
SC = O(h)

Flow of code



Hence size of largest BST is 3.

Q4 Merge 2 BST

1) Convert both BST to Linked list.
2) Merge 2 sorted linked list
3) Convert the linked list to BST.