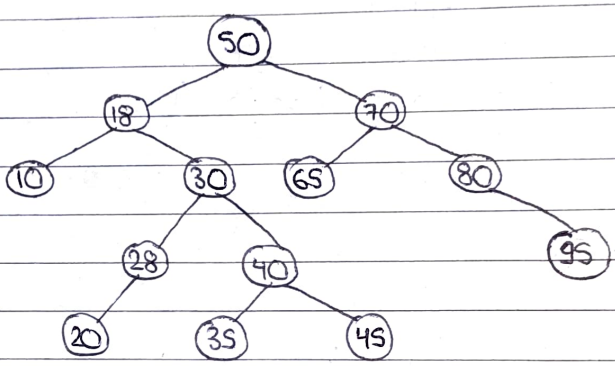
Date _____ Page _____

Day - 168

BST - 4

* Lowest Common Ancestor in a BST:



⇒ Suppose we have to find lowest common ancestor of 28 & 40 then the answer is 30.

⇒ In the simple approach, we can make a array that store all the nodes upto that node.

⇒ 50, 18, 30, 28, 20
50, 18, 30, 40

⇒ When we get any mismatch, we will stop & return the previous element.

⇒ But we have to solve it without using array.

Date _____ Page _____

⇒ As we can notice, we have BST, so we can find the side at which both the nodes are present.

⇒ So, here, at 50, both the nodes are present in left side.

⇒ Then for 18, at right side.

⇒ Then for 30, both are going in different direction.

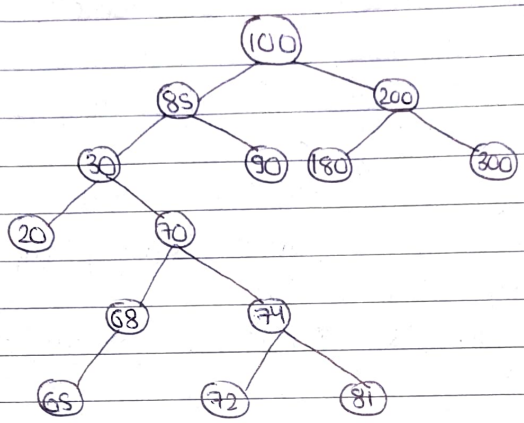
⇒ That means, 30 is the lowest common ancestor.

Code

```
Node * LCA( Node * root, int n1, int n2){
    if(!root)
        return NULL;
    if( root -> data > n1 && root -> data
                > n2){
        return LCA(root -> left, n1, n2);
    }
    else if( root -> data < n2 && root -> data
                < n2)
        return LCA(root -> right, n1, n2);
    else
        return root;
}
```

Date _____ Page _____

* Print BST element in given range:



⇒ We have to print all the elements in the given range in the sorted order.

⇒ In brute force approach, you can find the inorder and print the elements in that range.

⇒ Now, in 'mentos' approach, ~~you~~ we can use our previous question method.

⇒ At every node, check that node in the given range or not.

⇒ If yes, then go to left → print node → right.

⇒ Otherwise, go to left or right, according to the node.

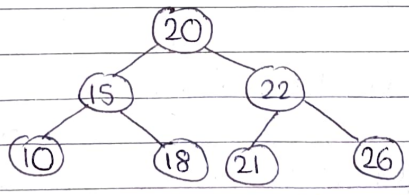
Date _____ Page _____

Code

```
void find(Node * root, vector<int> &ans, int &n1, int &n2){
    if(!root) return;

    if(root->data > n1 && root->data > n2)
        find(root->left, ans, n1, n2);
    else if(root->data < n1 && root->data < n2)
        find(root->right, ans, n1, n2);
    else{
        find(root->left, ans, n1, n2);
        ans.push_back(root->data);
        find(root->right, ans, n1, n2);
    }
}
```

* Check whether BST contains Dead End:



⇒ In this, we have to find that the tree have dead end or not that means if in any leaf node we can't add any ether node that means if means the tree contains dead end.

Date _____ Page _____

⇒ Here, 21 node is dead end because you can't any other node to it.
⇒ Also, this tree only contain unique nodes.

⇒ So, as we can see, we have to find a range.
⇒ So, we will start with 0 & INT_MAX.
⇒ Then go to left & right for checking.
⇒ If any node is not deadend then it will return 0.
⇒ If we get 1 then we will return 1.

⇒ For checking dead end node, we will do → (high - low) > 1 then return 0 that means not a dead end node & if (high - low) = 1 that means dead end node → return 1.

Code

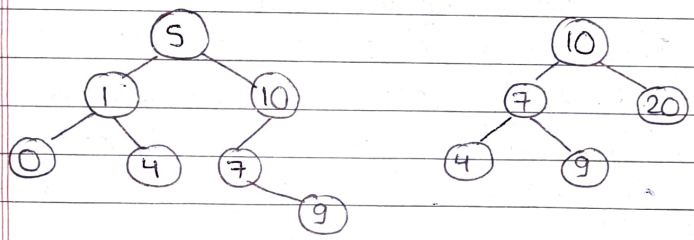
```
bool dead(Node *root, int lower, int
                                upper){
    if(!root)
        return 0;
    if(!root->left && !root->right){
        if(root->data - lower == 1 &&
            upper - root->data == 1)
            return 1;
        else
            return 0;
    }
```

Date _____ Page _____

return (Dead (root → left , lower ~~upper~~ root → data) || Dead (root → right , root → data , upper));

}

* Common node in two BST :



⇒ We have to return common nodes in both the BST.

⇒ Like as → 4 7 9 10

⇒ By using brute force, find the inorder & then check for same nodes & return that same nodes.

⇒ We will find inorder in a different way to optimize the approach.

⇒ First, we will push all the left most node in the stack.

⇒ Then pop that element & print it.

⇒ Then push the right element and after all the leftmost element of this push element.

Date _____ Page _____

⇒ Because the order is — L N R.

⇒ This technique, we will use for this solving the question.

⇒ So we will take two stack & push the leftmost elements to it,

⇒ Then start checking, if same then add to the answer otherwise select the smallest top element → pop it → push its right then leftmost of right.