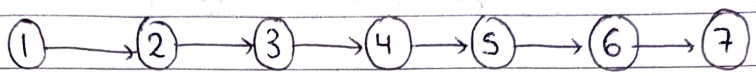
Date _____ Page _____

Day - 169

BST - 5

* Sorted LL to BST:

⇒



⇒ We have convert this LL to a balanced BST.

⇒ We will first find the mid then left side elements goes to left side & right side to right side. (we will make a node of the mid element then we will do above steps).

⇒ But if we directly use LL then our T.C. can be increased.

⇒ So, we can make a vector that store all the elements of the LL.

Code
```
TNode* createBST(LNode *head){
vector<int> tree;
while(head){
    Tree.push_back(head->data);
    head = head->next;
}
return BuildBST(Tree, 0, Tree.size()-1);
}
```

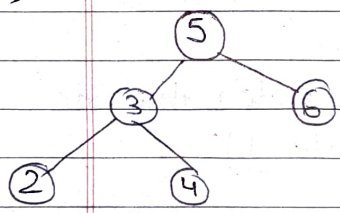
Date _____ Page _____

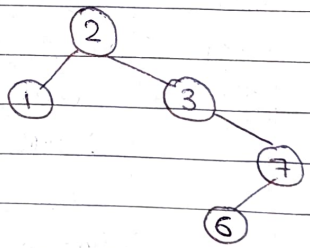
```
TNode * BuildBST(vector<int> &tree,
                 int start, int end){
    if(start > end)
        return NULL;
    int mid = (start+end+1)/2;

    TNode *root = newTNode(Tree[mid]);
    root->left = BuildBST(~~root~~ Tree,
                 start, mid-1);
    root->right = BuildBST(Tree, mid+1,
                 end);
    return root;
}
```

* Merge two BST:

⇒





⇒ We have to return a vector that contain all the elements of both the trees in sorted manner.

⇒ So, first we find the inorder of both the trees and store the result in the array.

Date _____ Page _____

Then create an array that contain all the elements in the sorted manner, for this we will use two variable & the smallest out of that element come in that big array.

Code

```
vector<int> mergeTwoBST(Node *root1,
                        Node *root2){
    vector<int> ans1;
    vector<int> ans2;
    inorder(root1, ans1);
    inorder(root2, ans2);

    vector<int> ans;
    int i=0, j=0;

    while(i<ans1.size() && j<ans2.size()){
        if(ans1[i] < ans2[j])
            ans.push_back(ans1[i++];
        else
            ans.push_back(ans2[j++]);
    }
    while(i<ans1.size())
        ans.push_back(ans1[i++]);
    while(j<ans2.size())
        ans.push_back(ans2[j++]);
    return ans;
}
```

Date _____ Page _____

```
void inorder (Node * root, vector<int>&ans)
    if(!root)
        return;
    inorder (root → left , ans);
    ans.push_back (root → data);
    inorder(root → right, ans);
}
```

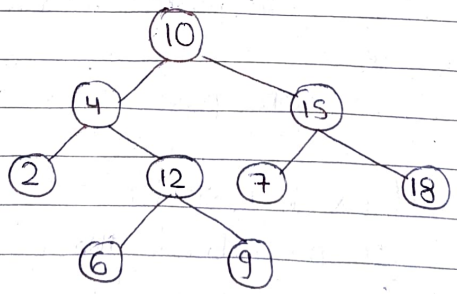
⇉ How to solve this question by iterative method.

⇉ For this, we will use inorder iterative method.

⇉ we will take two stacks, then do all the inorder steps.

⇉ First, push all the elements then go the right & push all the left element of the right & after that do compiarision

* Fixing two Nodes of a BST:



⇉ There are two nodes that swapped.

Date _____ Page _____

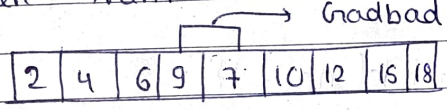
⇒ We have to find the nodes and swapped that.

⇒ In the simple approach, we can find inorder then sort that array & again fill all the element of the array to the tree.

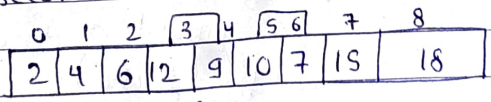
⇒ ~~We can~~ For more optimized approach, we can find that two elements then swapped it.

⇒ This will take O(N) T.C & O(N) S.C.

⇒ Now, if we get 1 Badbad then swap between them i.e.



⇒ If we get 2 Badbad then swap first element of 1st Badbad with 2nd element of second Badbad.



⇒ swap 12 & 7.

⇒ we will take 2 variables first & second.

⇒ Init with -1.

⇒ when the Badbad is -1 of first that means this is first Badbad.

Date _____ Page _____

⇉ If the second is -1 then it is second gadbad.

⇉ What if we have to solve this question in O(1) S.C.

⇉ we will use morris traversal for this.

⇉ we will take two variables prev & curr.

⇉ If the prev value is greater than curr value then it is a gadbad.

Code

```
void correctBST(Node * root){
    Node * curr = NULL;
    Node * First = NULL, * second = NULL;
    Node * last = NULL, * present = NULL;

    while(root){
        if(!root → left){
            last = present;
            present = root;
            if(last && last → data >
                present → data){
                if(!first)
                    first = last;
                else
                    second = present;
            }
            root = root → right;
        }
    }
```

Date _____ Page _____

```
else{
    curr= root->left;                    !=root
    while(curr->right && curr->right!=root)
        curr = curr->right;
    if(!curr->right){
        curr->right = root
        root = root->left;
    }
    else{
        curr->right = NULL;
        last = present;
        present = root;
        if( last && last->data > present
                ->data){
            if(! first)
                first = last;
            second = present;
        }
        root = root->right;
    }
}
int num = first->data;
first->data = second->data;
second->data = num;
}
```