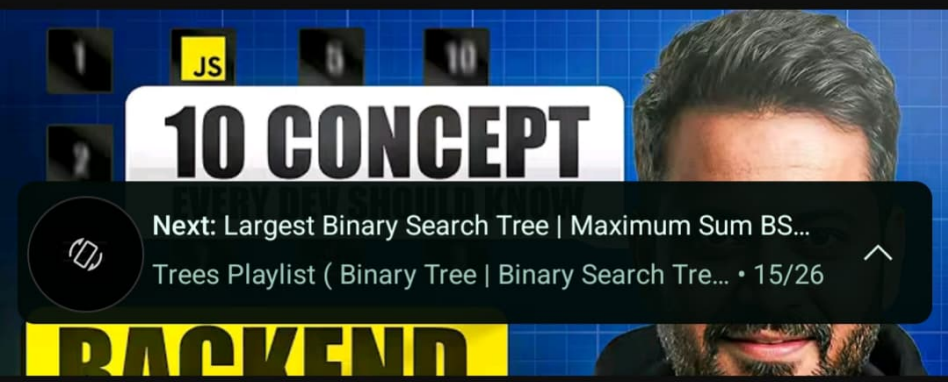

All | From the series | From Coder Army | Computer pro

Watch | Learn more


Largest Binary Search Tree | Maximum Sum BST in Binary Tree | Leetcode

Coder Army · 15K views · 1 year ago




Next: Largest Binary Search Tree | Maximum Sum BS...

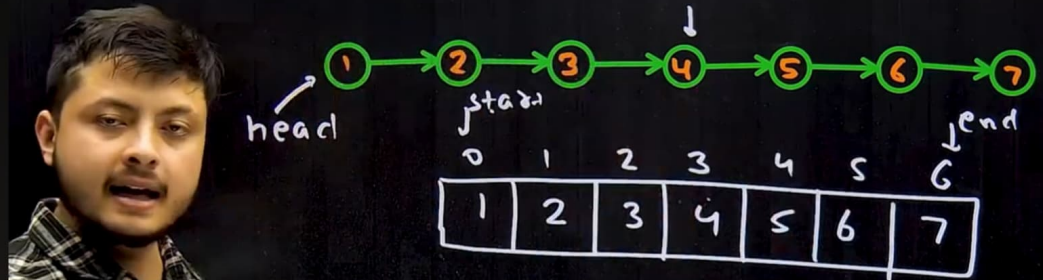
Trees Playlist ( Binary Tree | Binary Search Tre... • 15/26

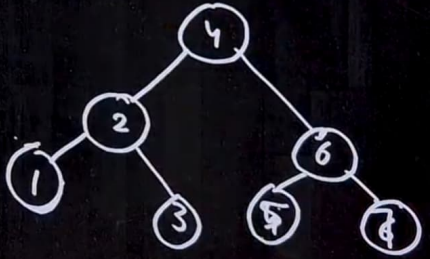
# Sorted LL to BST

# Sorted LL to BST

O(N)
O(N)

# Sorted LL to BST

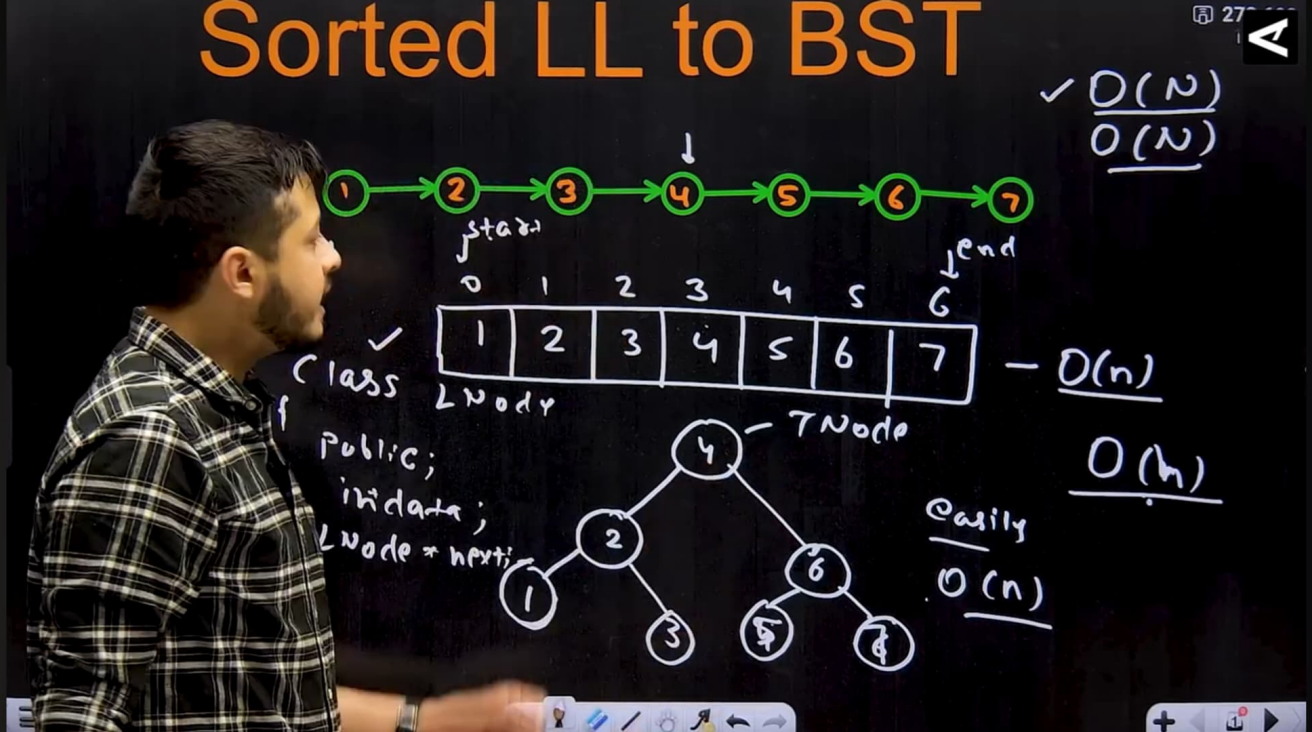


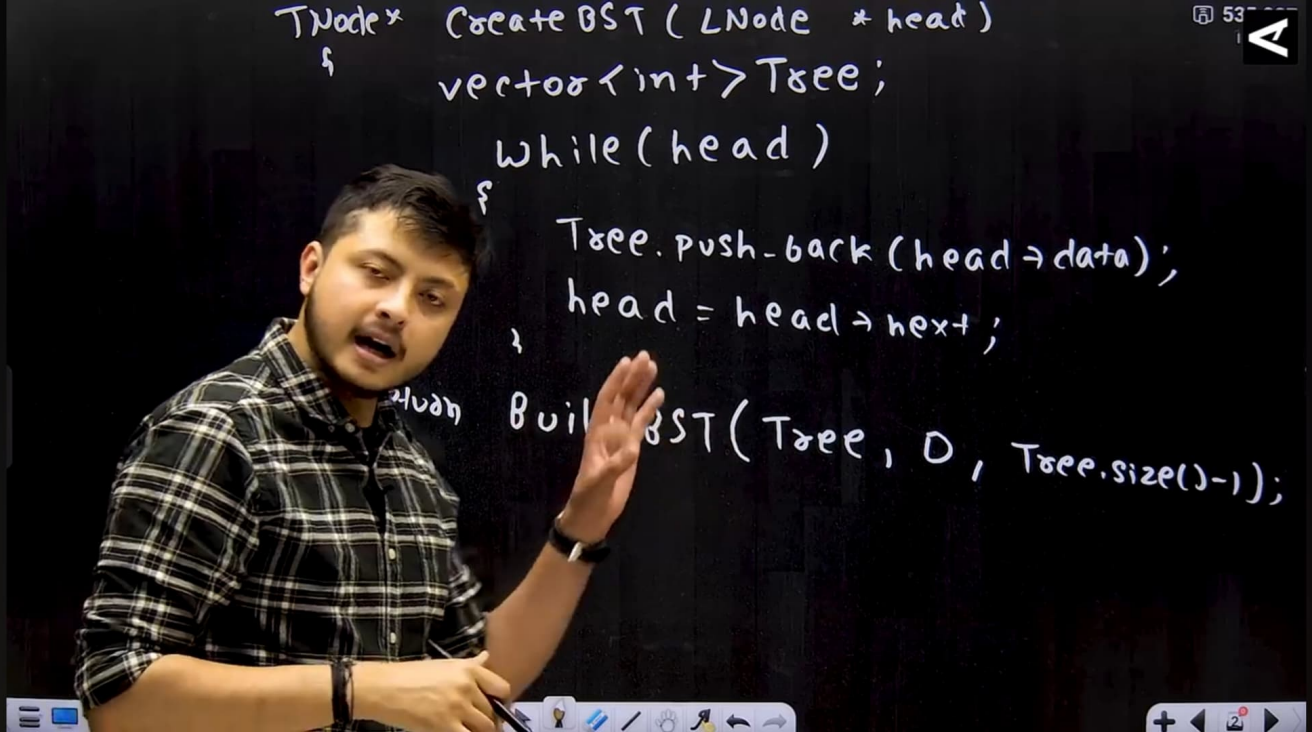



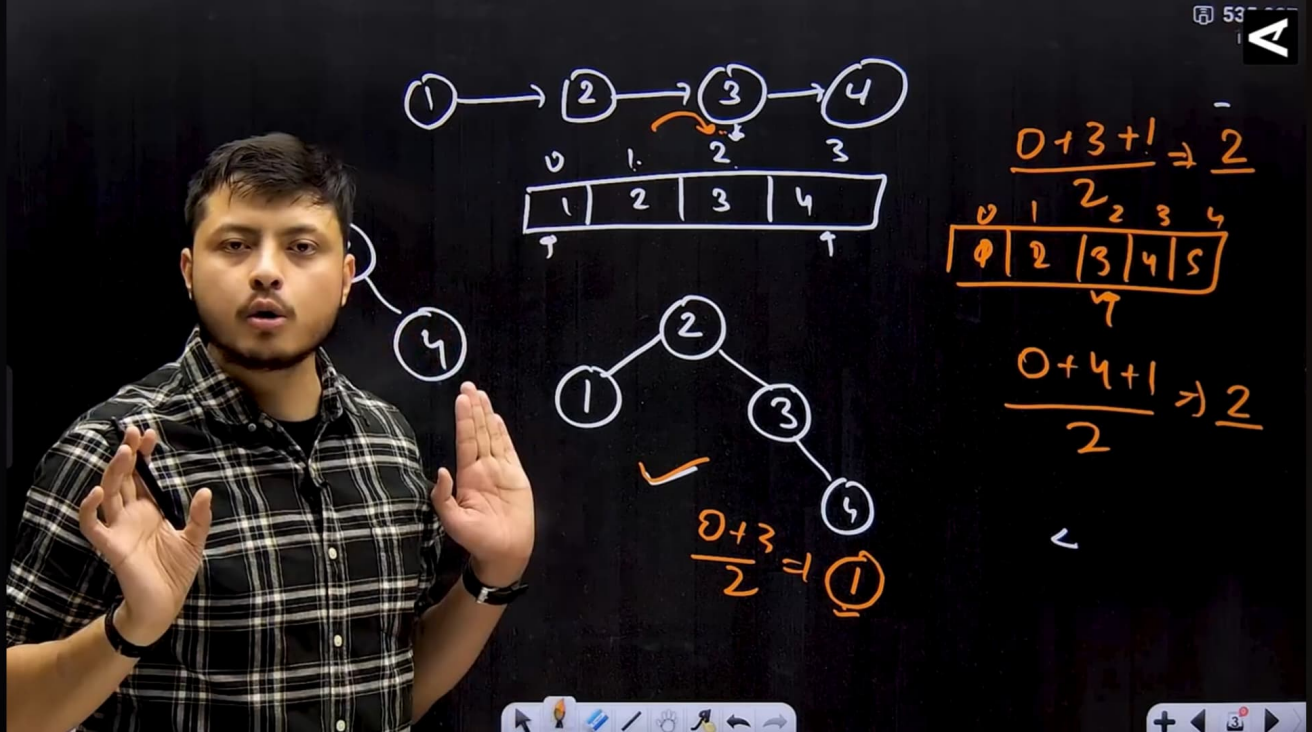

O(N)
O(N)

easily
O(n)

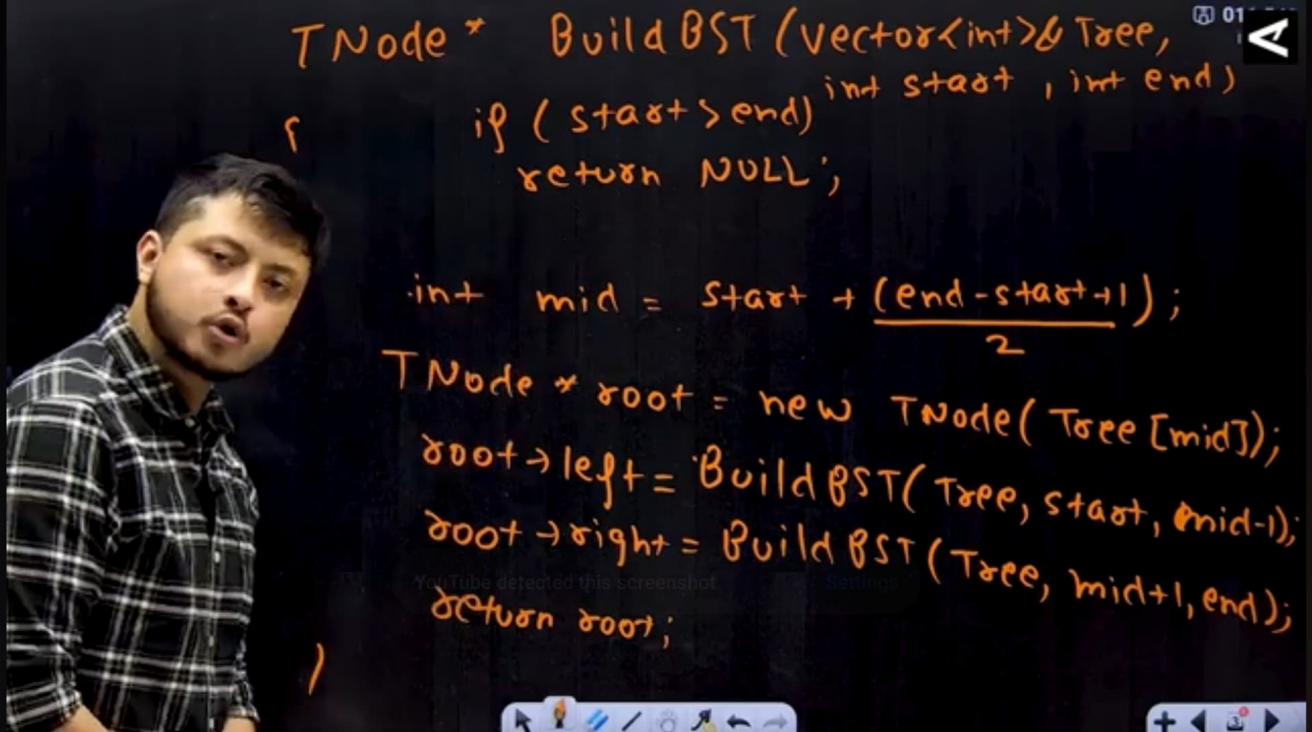
# Sorted LL to BST



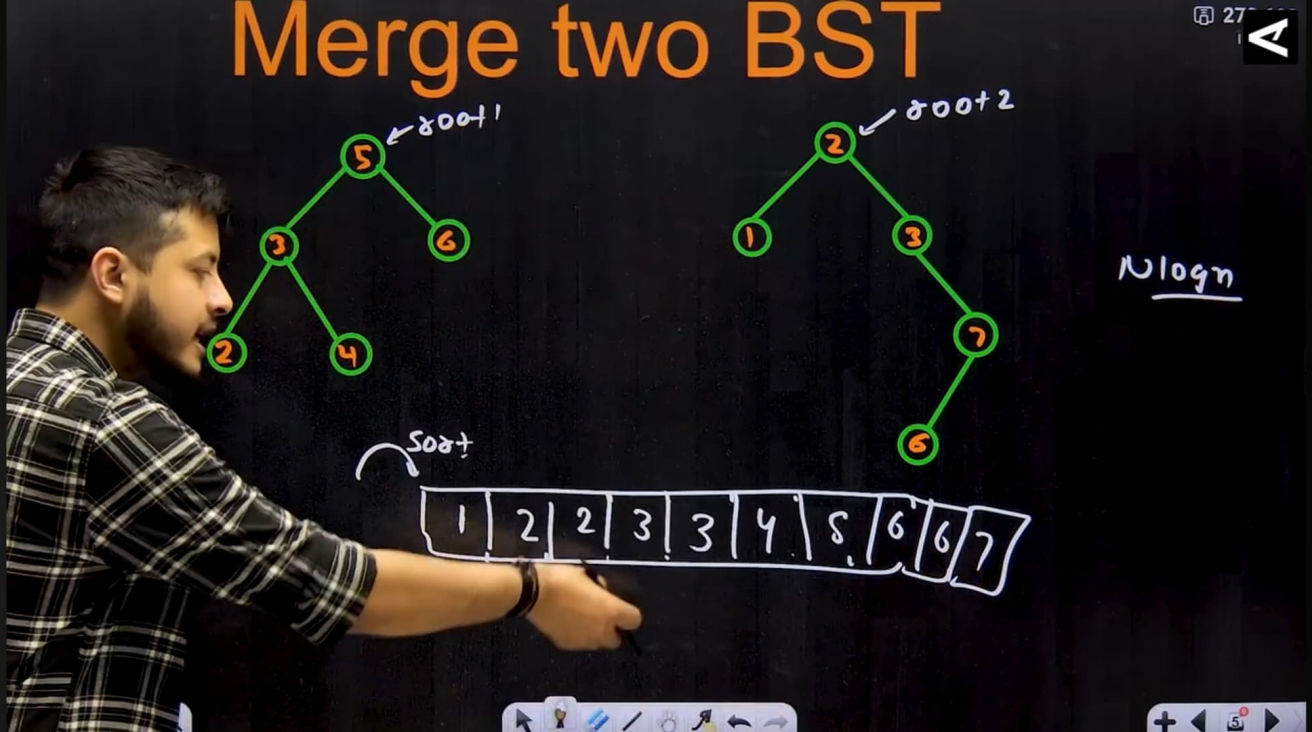

TNode* CreateBST(LNode *head)
{
vector<int> Tree;
while(head)
{
Tree.push_back(head->data);
head = head->next;
}
BuilBST(Tree, 0, Tree.size()-1);


1 2 3 4
0 1 2 3
1 2 3 4
0+3+1 / 2 → 2
0 1 2 3 4
0 2 3 4 5
0+4+1 / 2 → 2
2
1 3 4
0+3 / 2 → 1

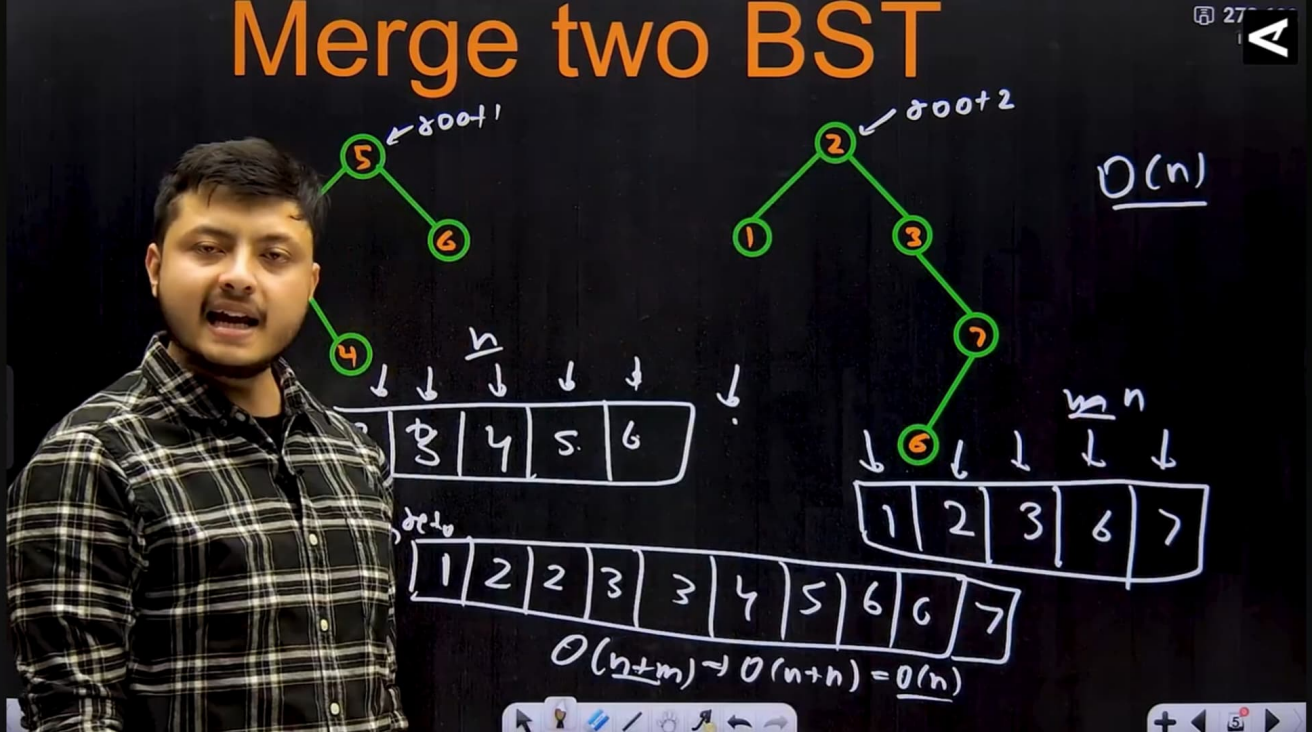
```
TNode* BuildBST(vector<int>& Tree, int start, int end)
{
    if (start > end)
        return NULL;

    int mid = (start + end + 1) / 2;   // Integer overflow

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

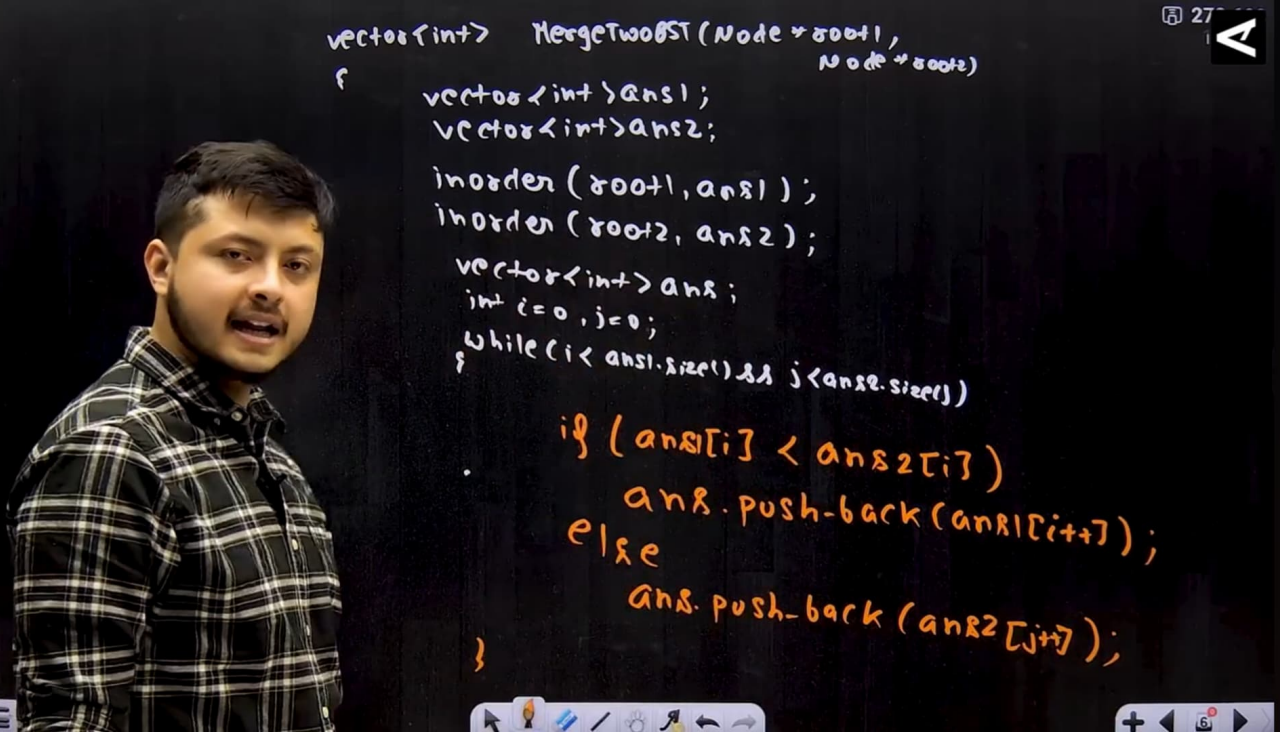

TNode * BuildBST(vector<int>& Tree, int start, int end)
{
if (start > end)
return NULL;
int mid = start + (end - start + 1) / 2;
TNode * root = new TNode(Tree[mid]);
root->left = BuildBST(Tree, start, mid-1);
root->right = BuildBST(Tree, mid+1, end);
return root;
}

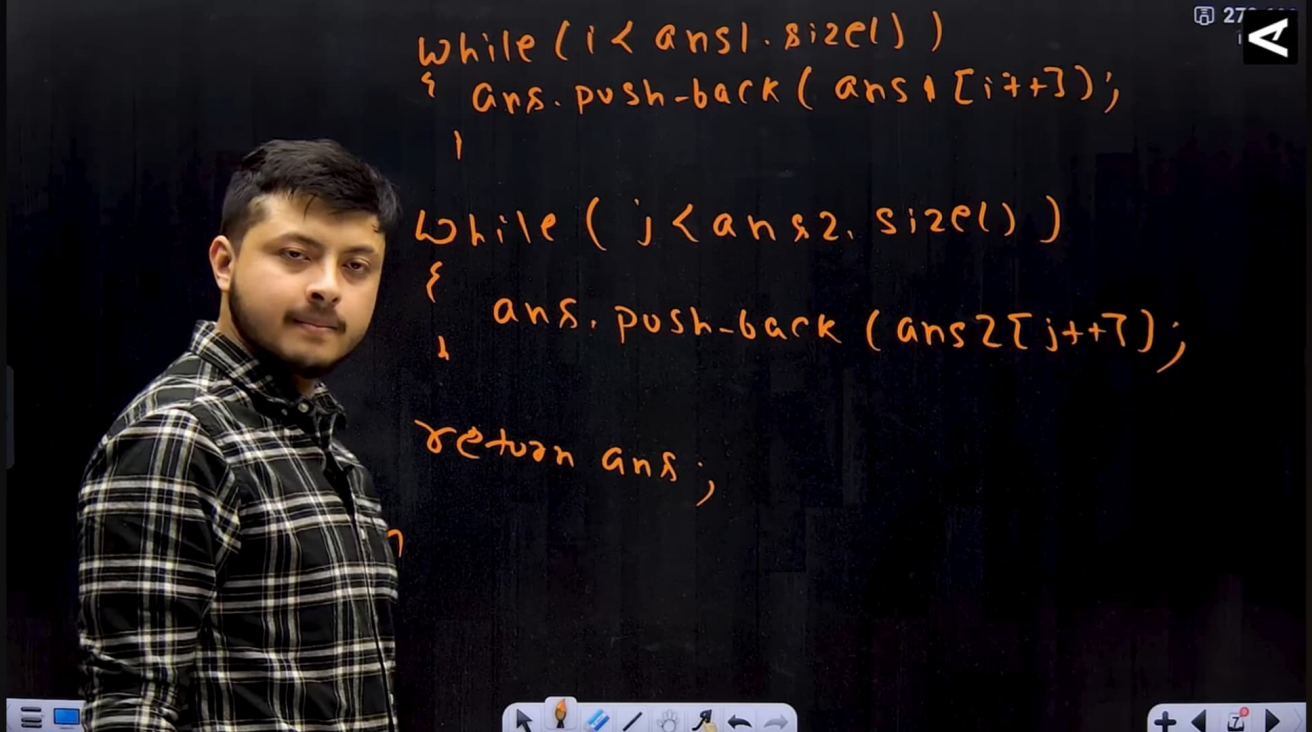
# Merge two BST



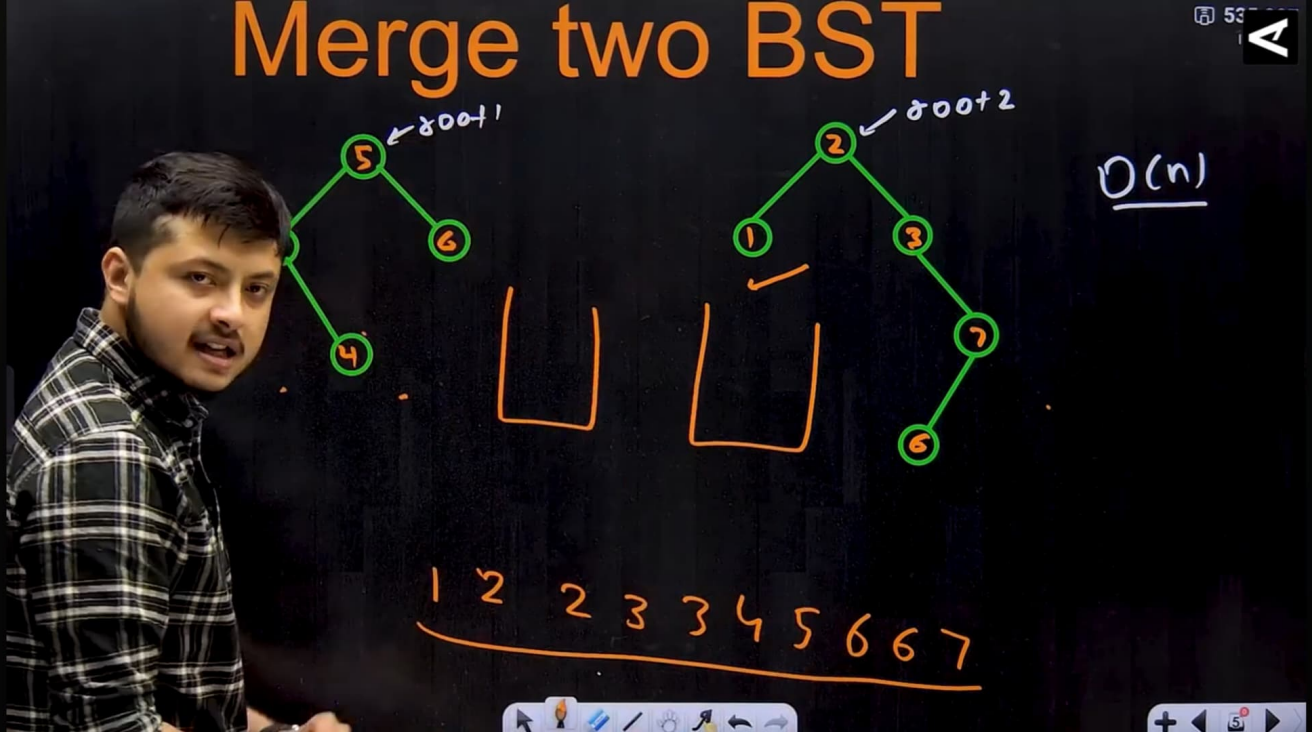
# Merge two BST



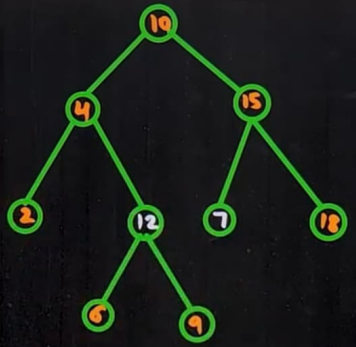

vector<int> MergeTwoBST(Node *root1, Node *root2)
{
vector<int>ans1;
vector<int>ans2;
inorder(root1,ans1);
inorder(root2, ans2);
vector<int>ans;
int i=0,j=0;
while(i<ans1.size()&& j<ans2.size())
{
if(ans1[i] < ans2[j])
ans.push_back(ans1[i++]);
else
ans.push_back(ans2[j++]);
}


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

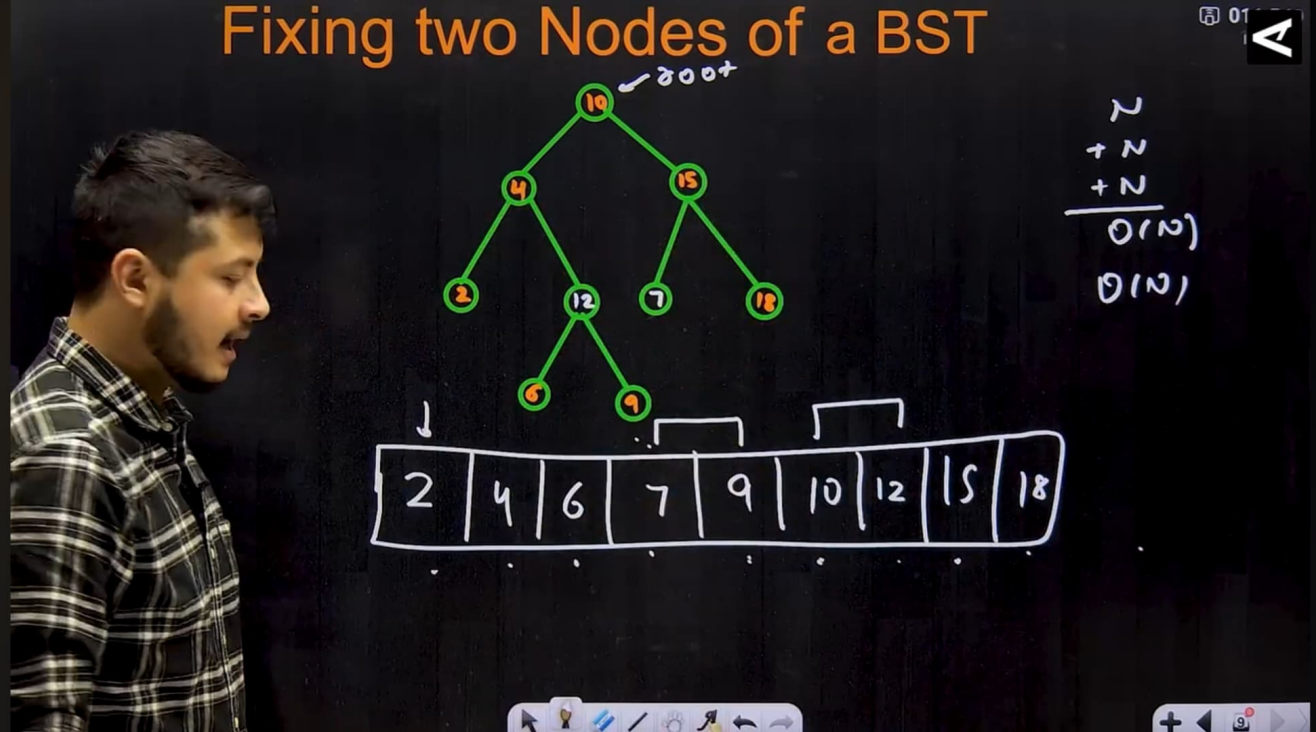
# Merge two BST



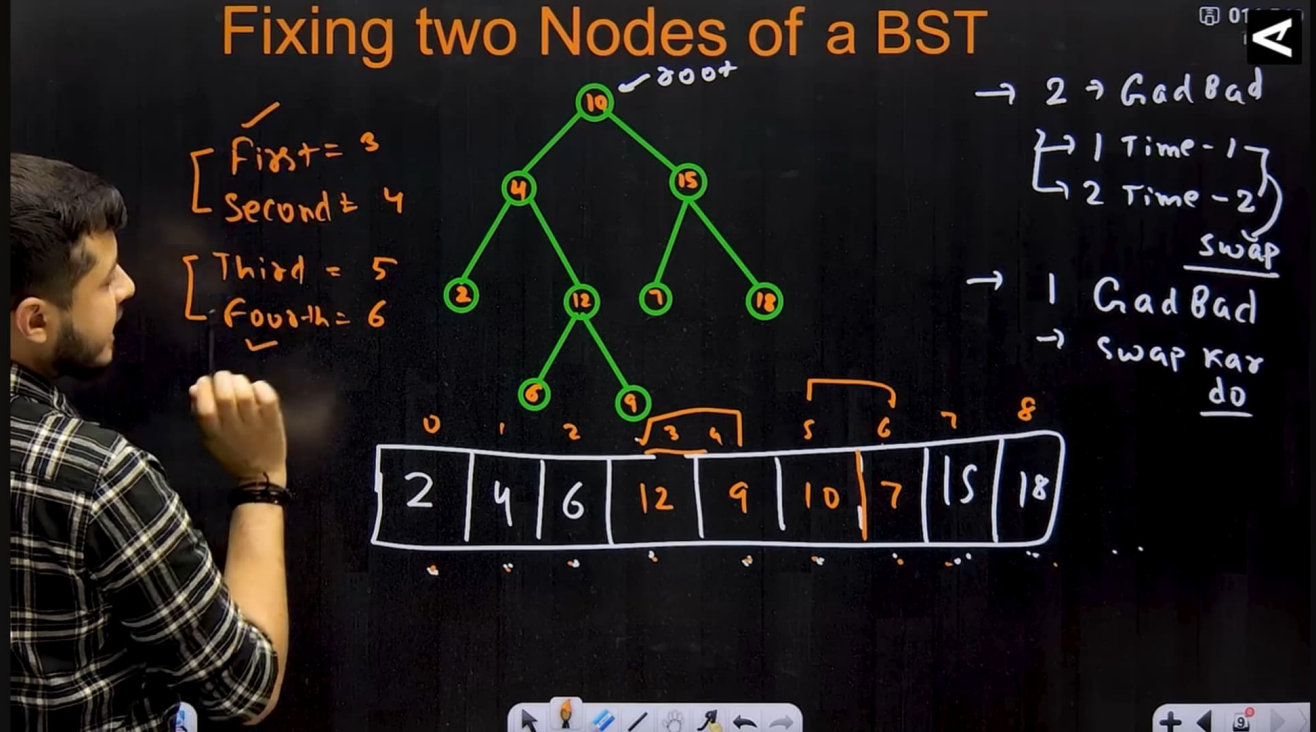
# Fixing two Nodes of a BST



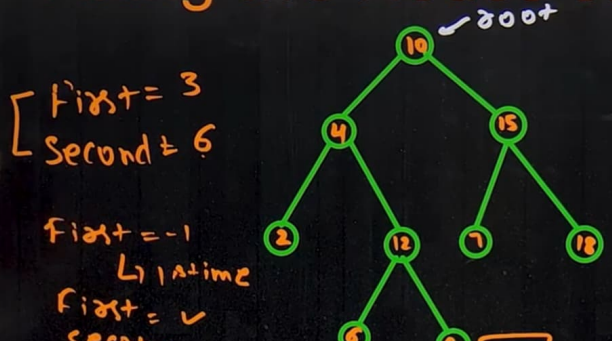
# Fixing two Nodes of a BST



# Fixing two Nodes of a BST



# Fixing two Nodes of a BST




First = 3
Second = 6

First = -1
↳ 1st time
First = ✓
seron = ✓

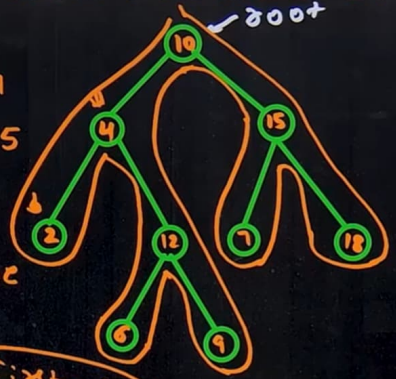
First != -1
↳ second = ✓

→ 2 → Gadbad
↳ 1 Time - 1
↳ 2 Time - 2
swap

→ 1 Gadbad
→ swap kar do

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 12 | 9 | 10 | 7 | 15 | 18 |

# Fixing two Nodes of a BST




First = 4
Second = 5

First = -1
↳ 1st time
First = ✓
Second = ✓

First != -1
↳ Second = ✓

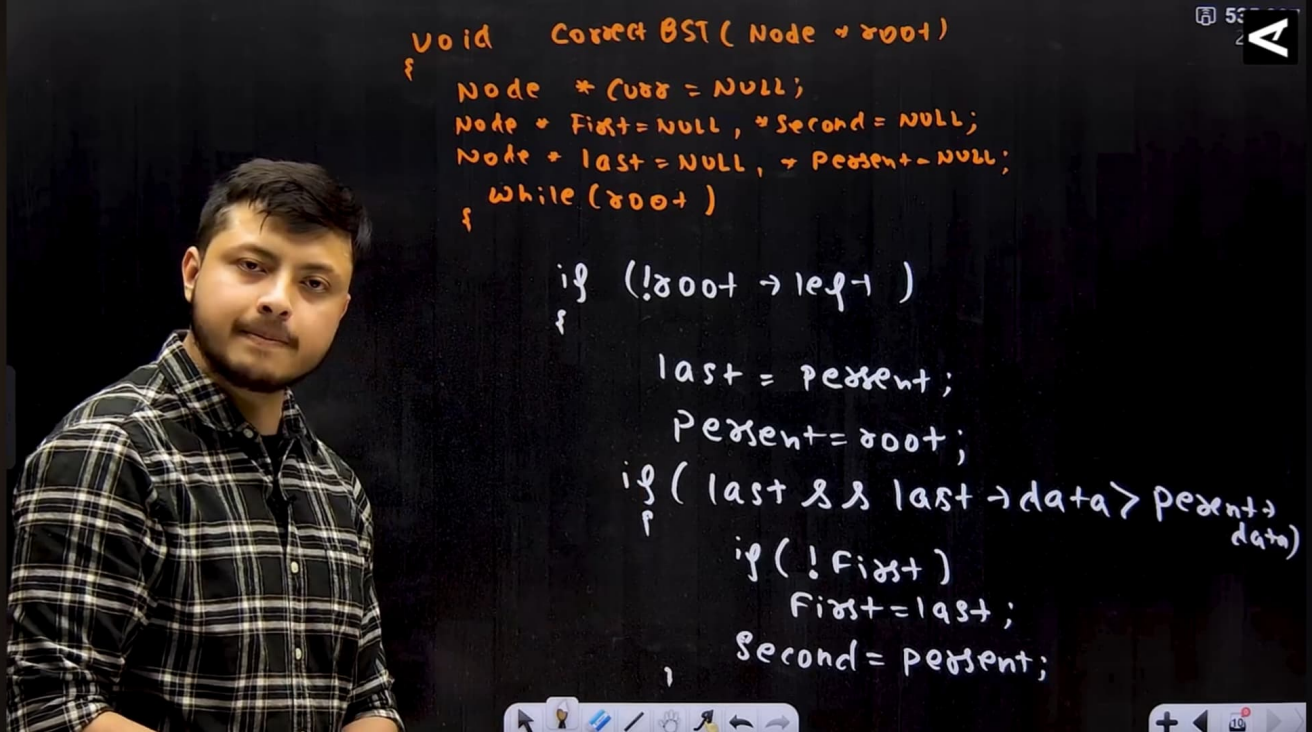
First = 12 ✓
Second = ✗

Curr = 4 6 12 9 10 7 15
Prev = 4 10 7 15

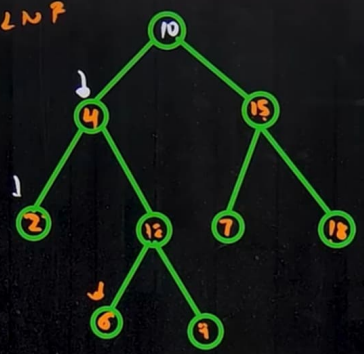
O(N)
O(N)
O(1)

→ 2 → Gadbad
↳ 1 Time - 1
↳ 2 Time - 2
swap

1 Gadbad
→ swap kar do


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;
Second = present;
}

# Fixing two Nodes of a BST




LNR

First = 4
Second =

First = -1
↳ last
First = ✓
secon = ✓

First != NULL ✓
↳ second = ✓

last = NULL
present = 2

→ 2 → Gadbad

O(N)
O(N)
O(1)

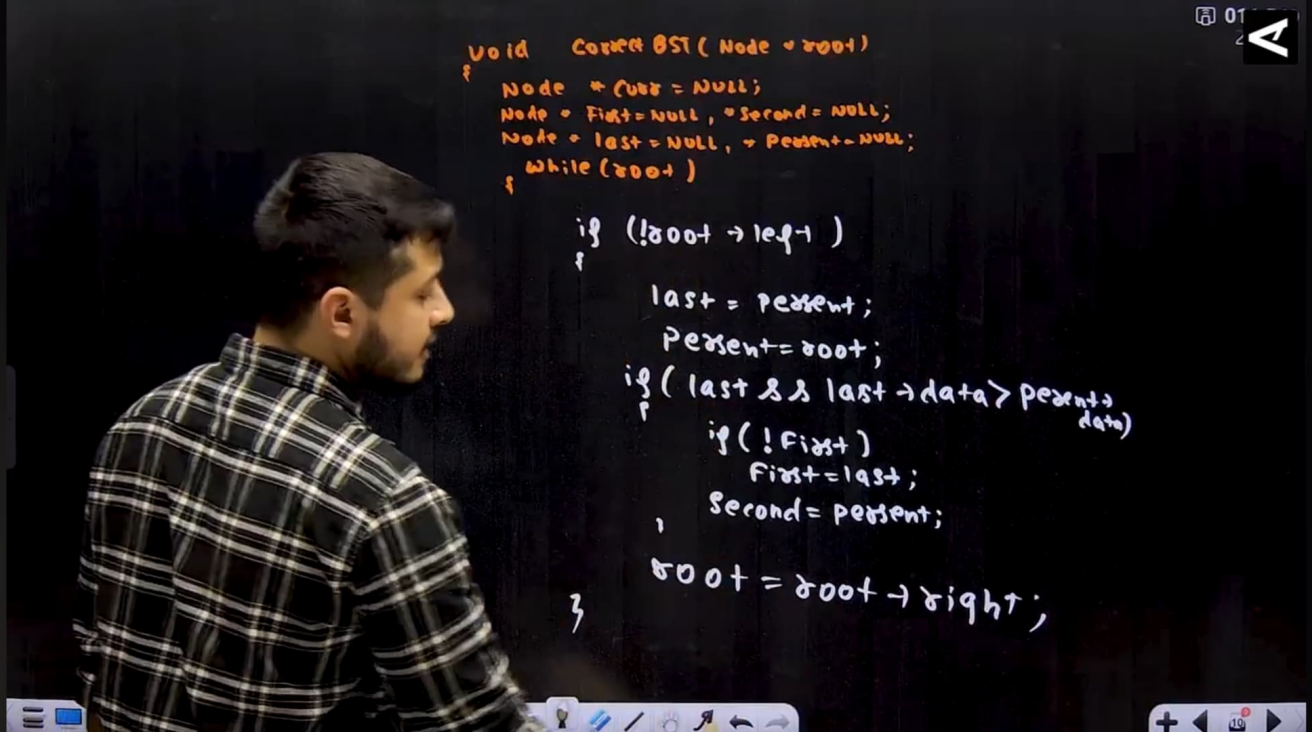
↳ 1 Time - 1
↳ 2 Time - 2
swap

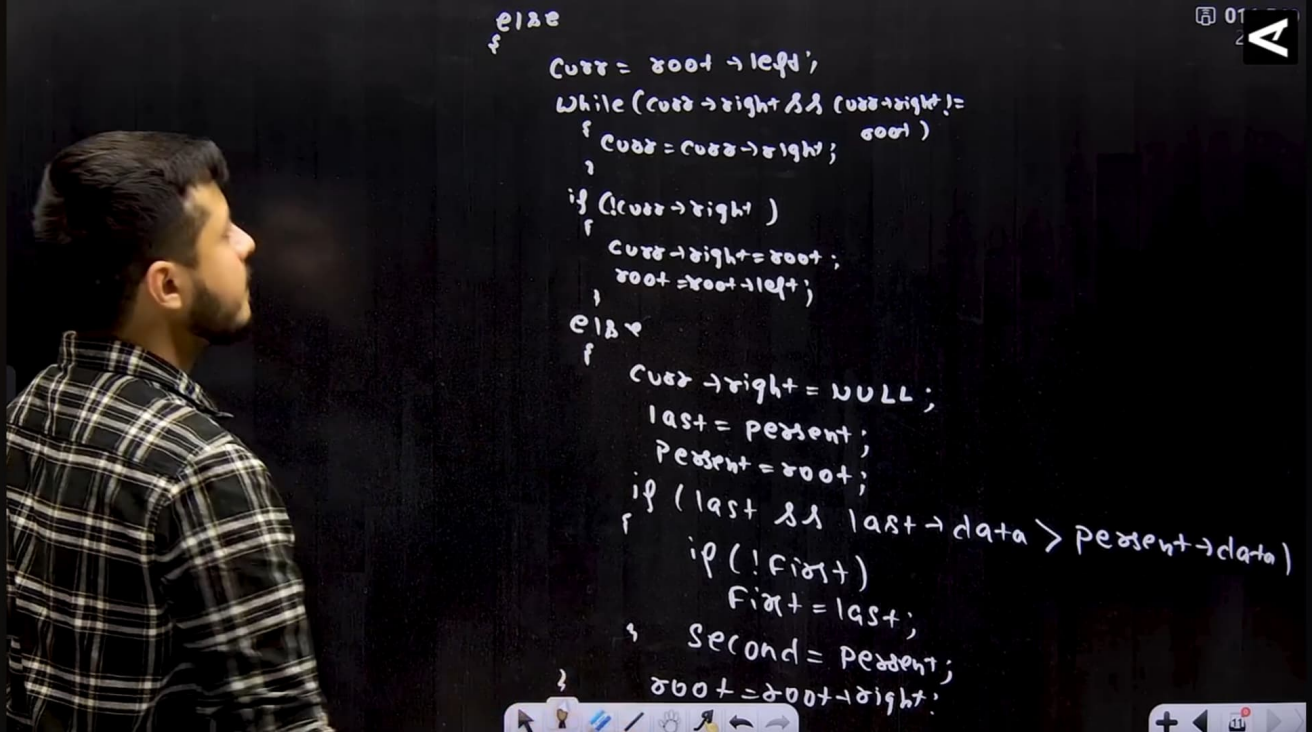
→ 1 Gadbad
→ swap kar do

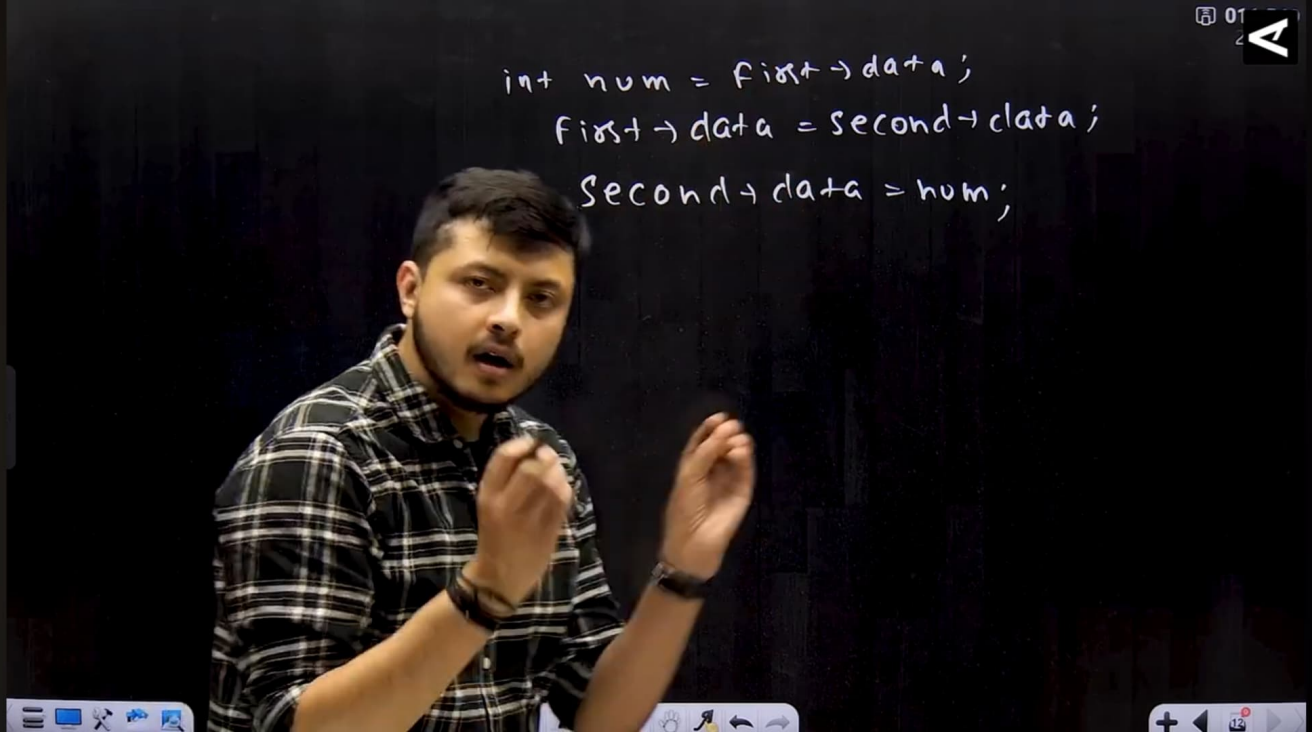
LNR

left doesn't exist
↳ node
↳ Right

left side exist
↳ Already traverse
↳ Not traverse


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;
second= present;
}
root = root->right;
}


else
{
curr = root->left;
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;