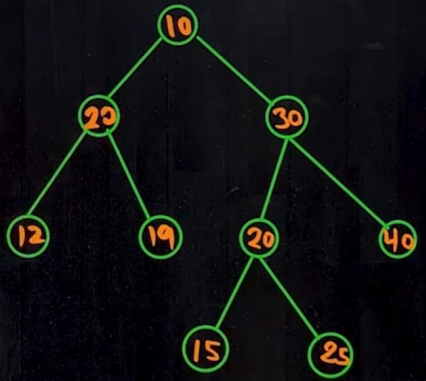
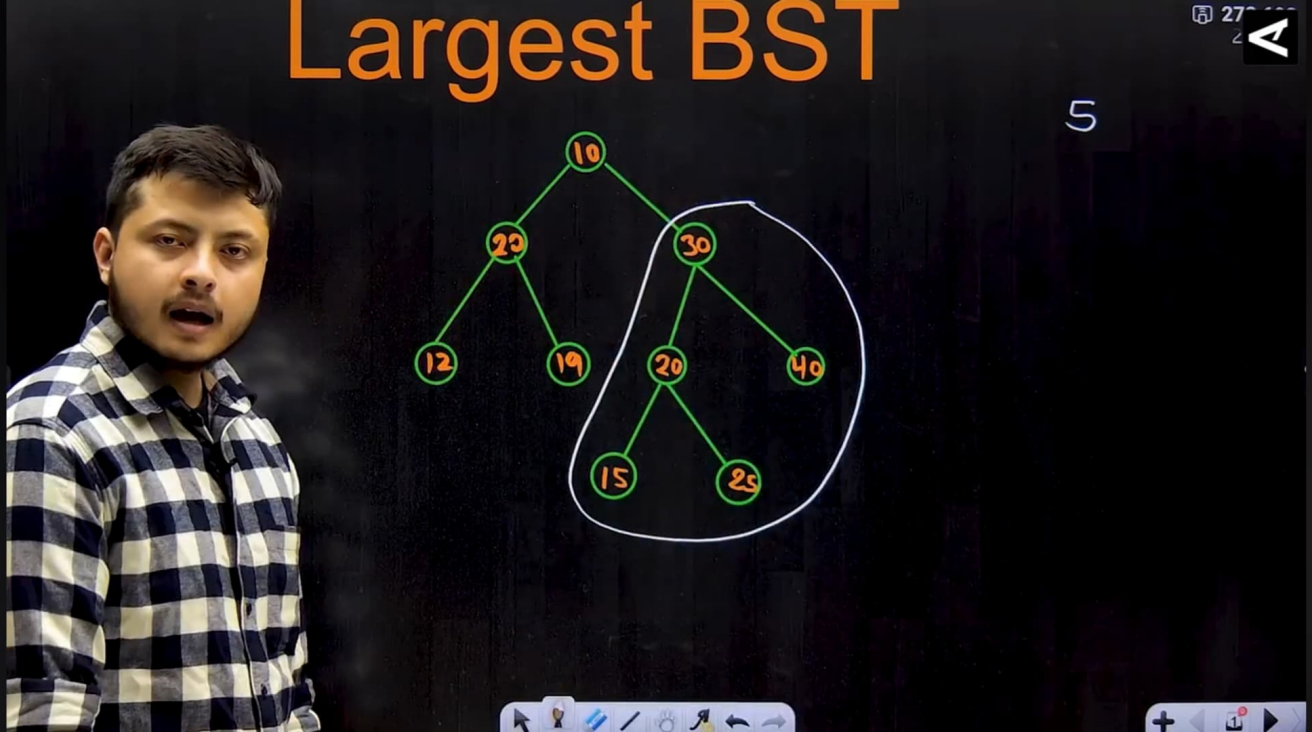
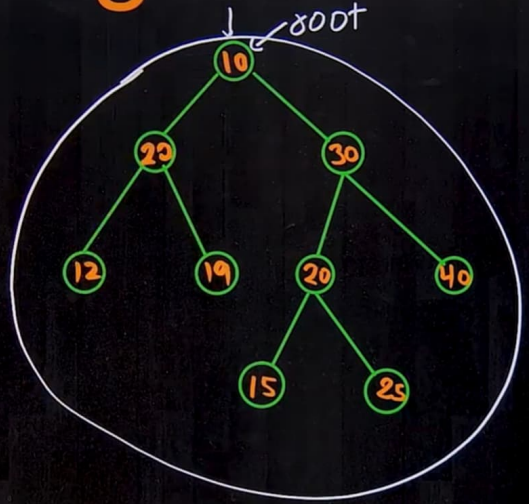
Largest BST



Largest BST



Largest BST

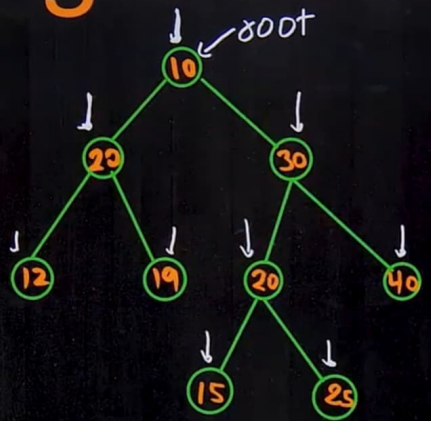


JSBST

no of node:

Largest BST

Largest BST



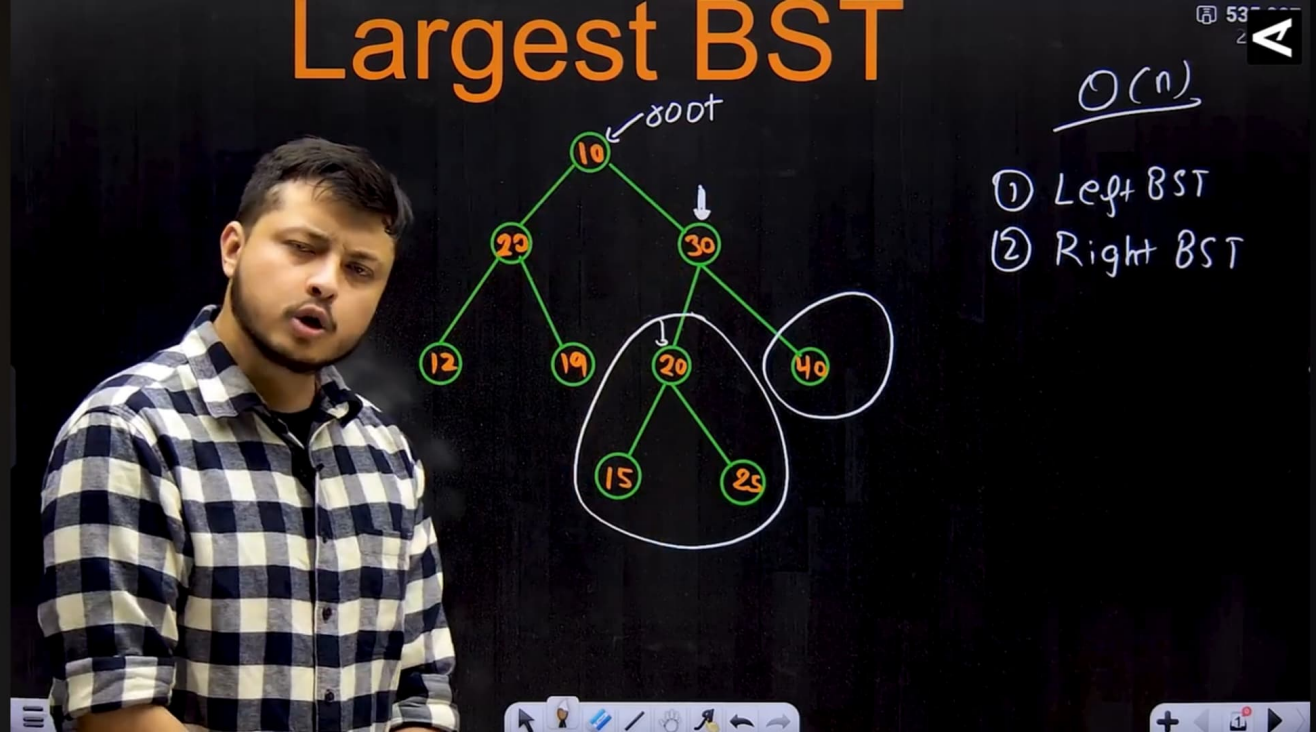
Size = 5

IS BST

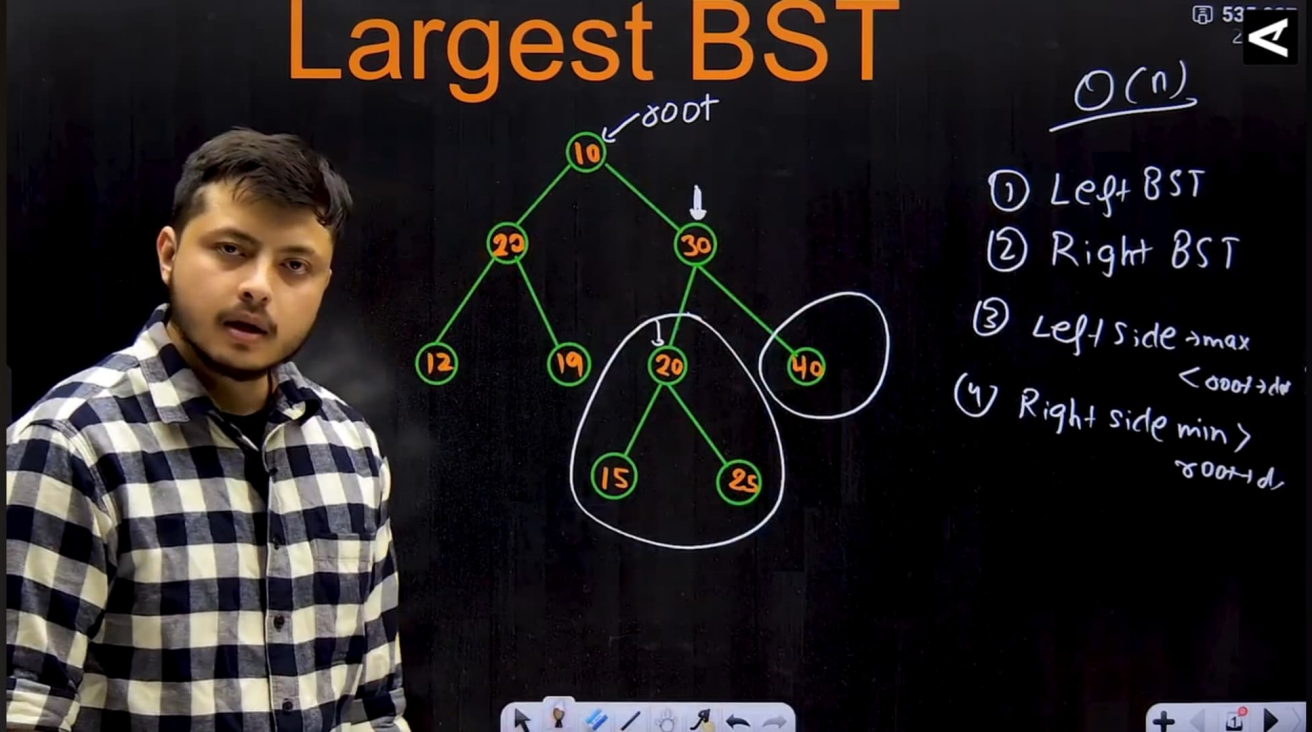
$h \times O(n)$

$O(n^2)$

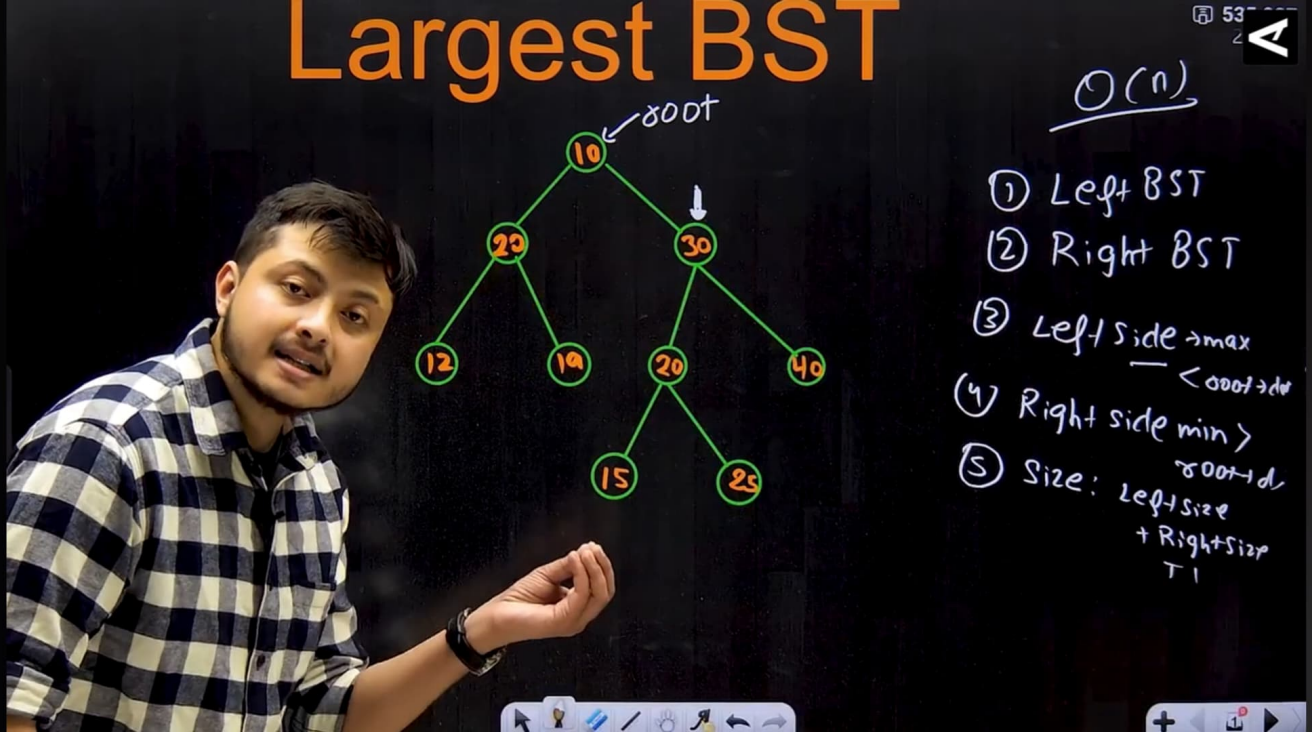
Largest BST



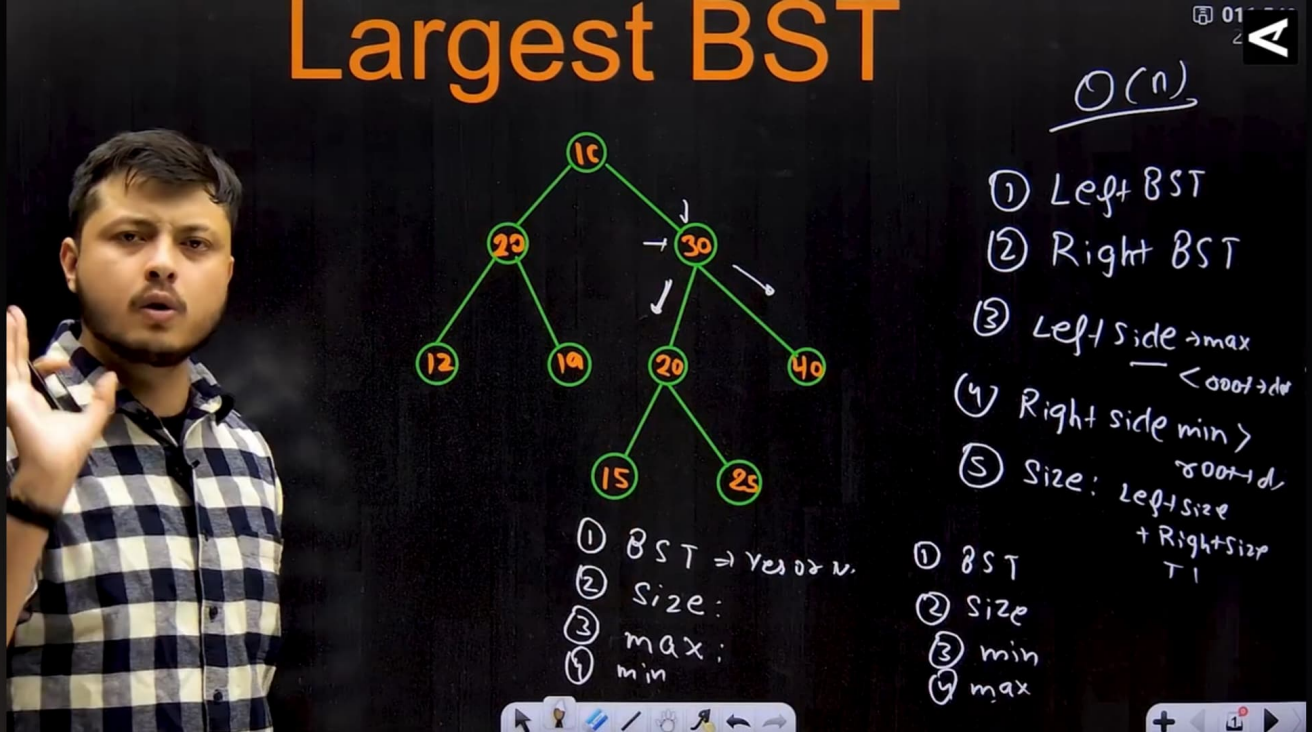
Largest BST


root
10
20
30
12
19
20
40
15
25
O(n)
① Left BST
② Right BST
③ Left side → max
< root → data
④ Right side min >
root → data

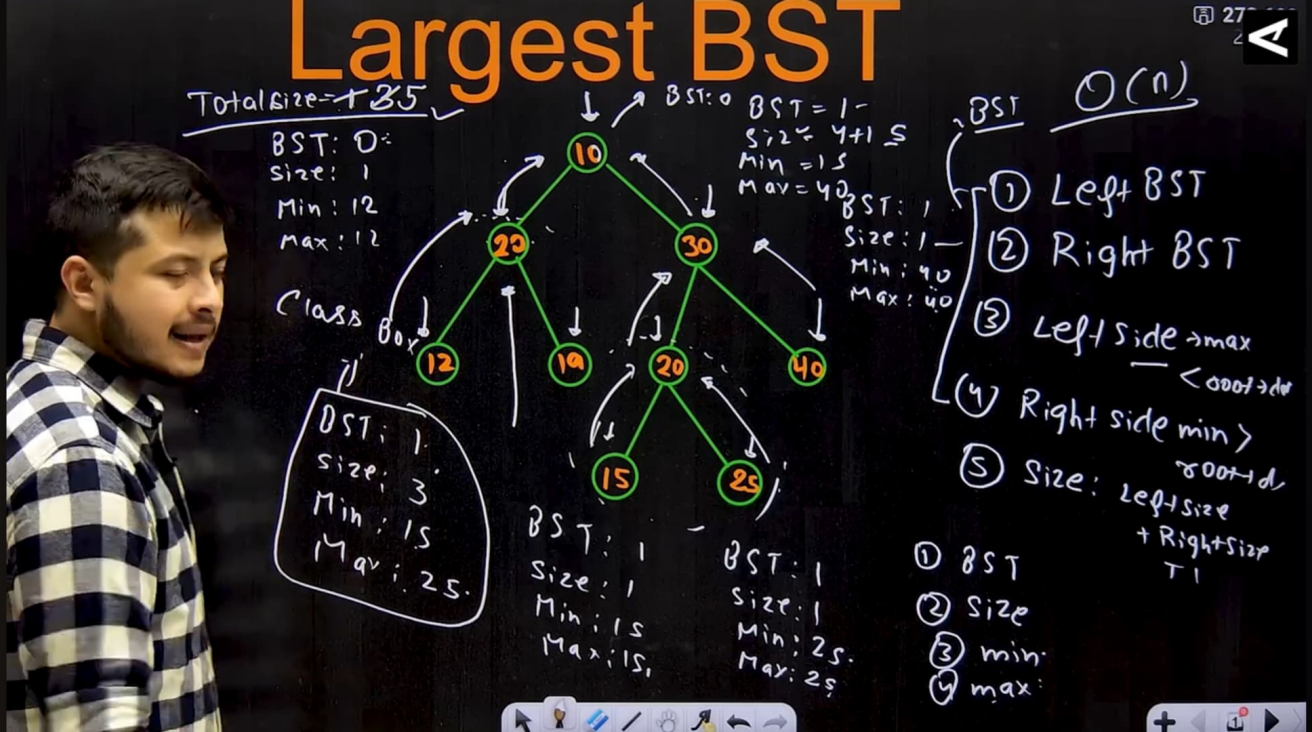
Largest BST

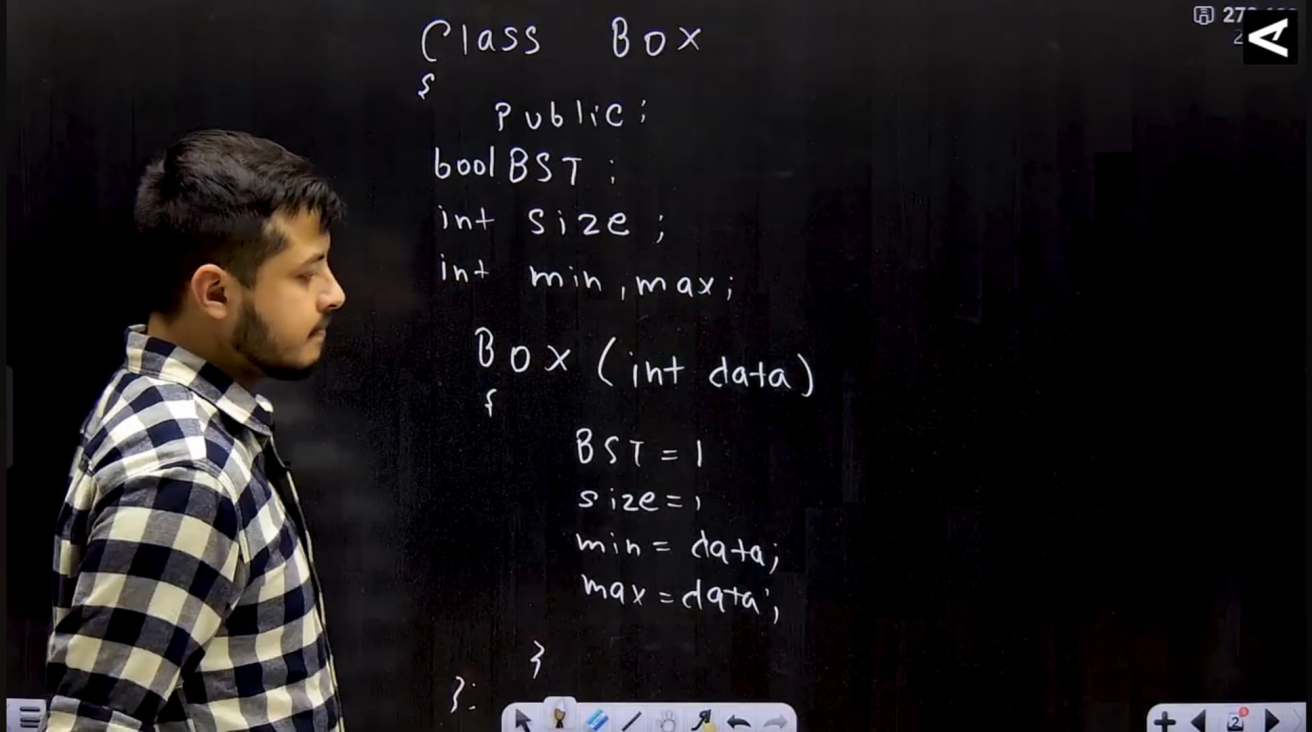

root
10
20
30
12
19
20
40
15
25
O(n)
① Left BST
② Right BST
③ Left side → max < root → data
④ Right side min > root → data
⑤ Size: Left Size + Right Size + 1

Largest BST


O(n)
10
20
30
12
19
20
40
15
25
① Left BST
② Right BST
③ Left side → max
< root → data
④ Right side min >
root → data
⑤ Size: Left Size
+ Right Size
+ 1
① BST ⇒ Yes or No.
② Size:
③ max;
④ min
① BST
② Size
③ min
④ max

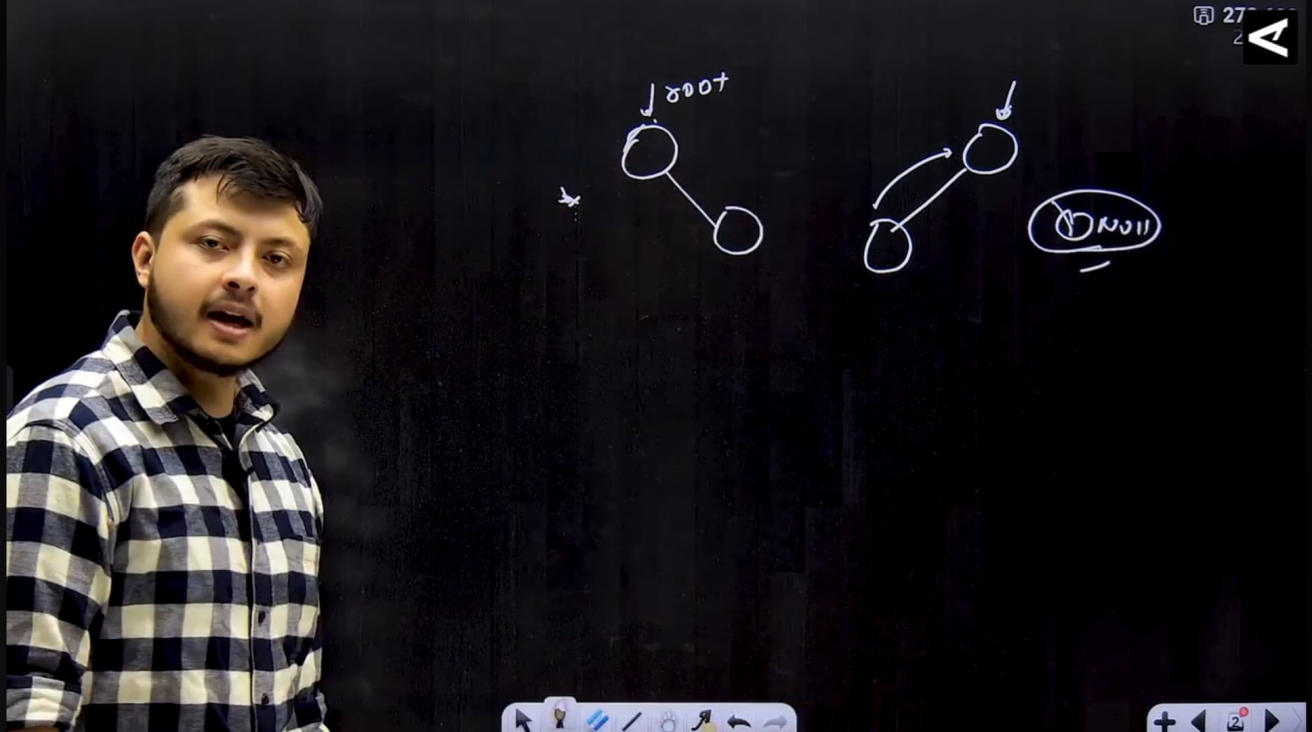
Largest BST

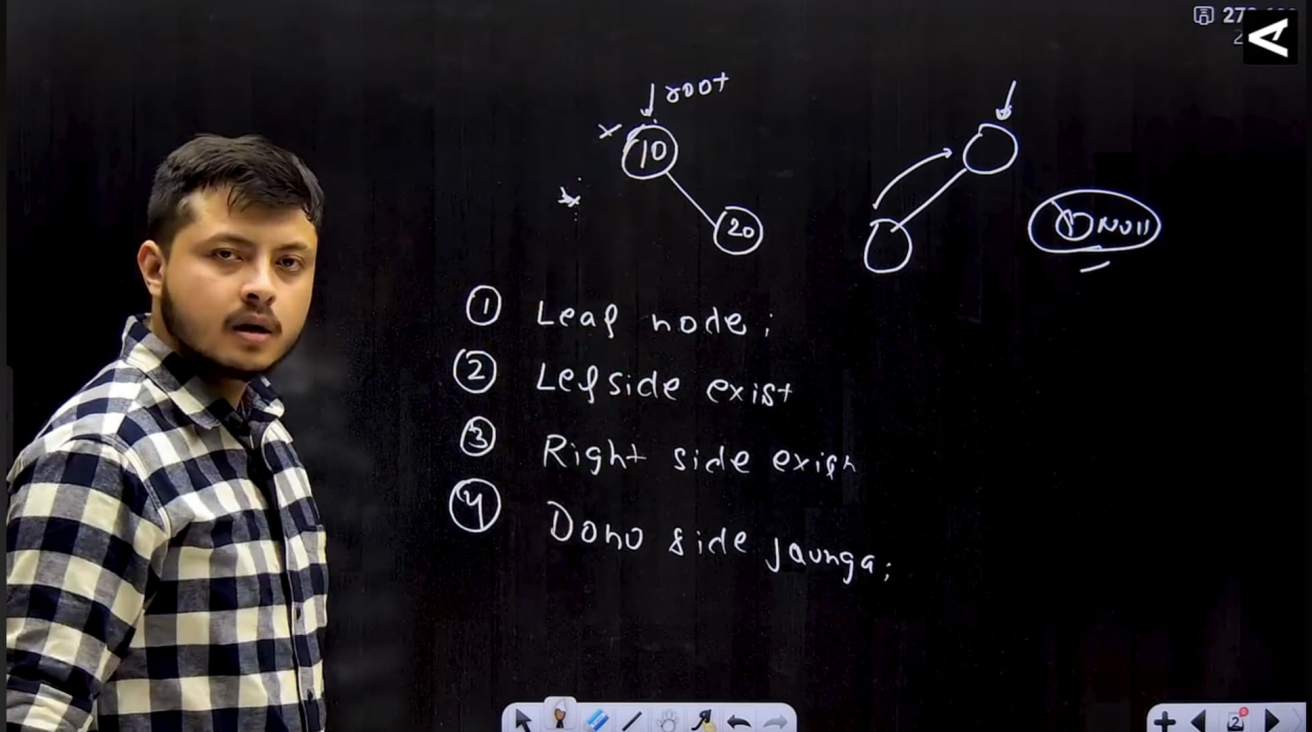

Total Size = 135
BST: 0
Size: 1
Min: 12
Max: 12
BST: 0
BST = 1
Size = 4+1 = 5
Min = 15
Max = 40
BST: 1
Size: 1
Min: 40
Max: 40
Class Box
BST: 1
Size: 3
Min: 15
Max: 25
BST: 1
Size: 1
Min: 15
Max: 15
BST: 1
Size: 1
Min: 25
Max: 25
10
20
30
12
19
20
40
15
25
BST O(n)
1. Left BST
2. Right BST
3. Left side → max < root → data
4. Right side min > root → data
5. Size: Left Size + Right Size + 1
1. BST
2. Size
3. min
4. max


Class Box
{
public:
bool BST;
int size;
int min, max;
Box(int data)
{
BST = 1
size = 1
min = data;
max = data;
}
};

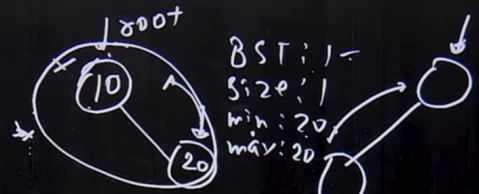
```
Box* Find ( Node *root , int &Totalsize)
{

}
```


root
NULL

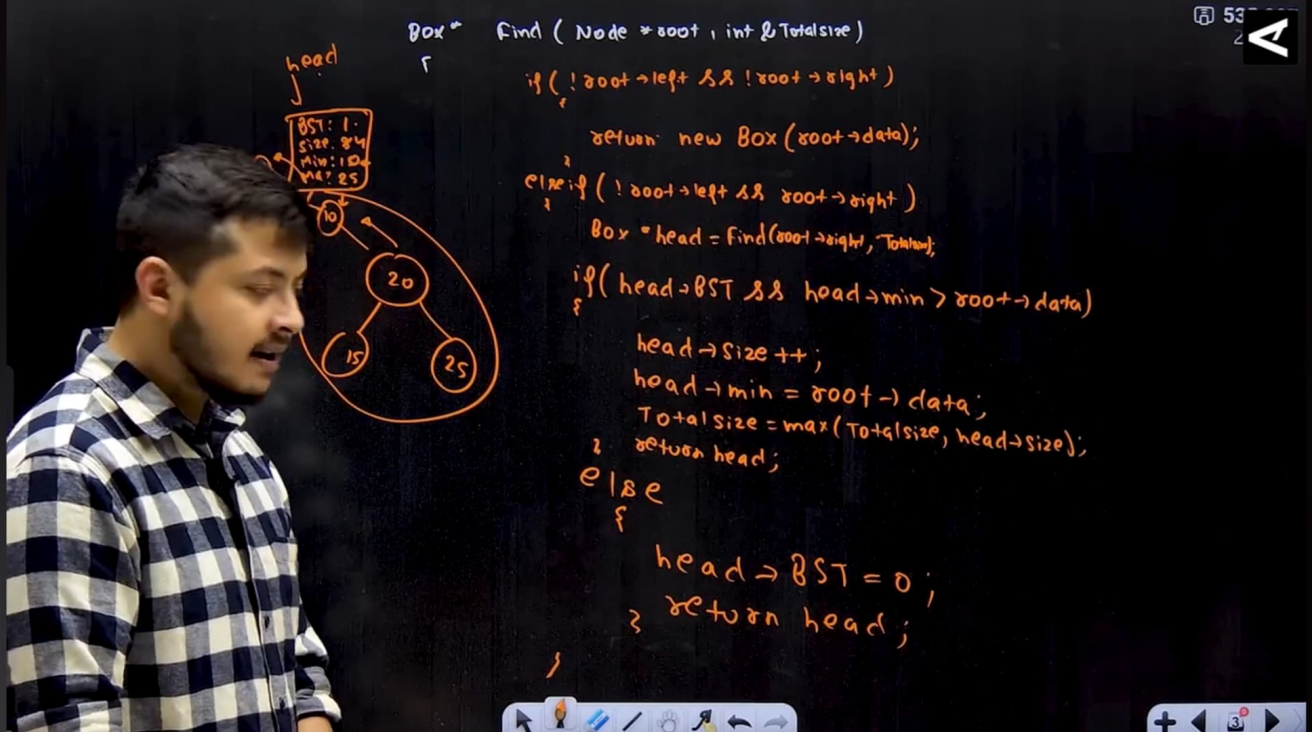

root
10
20
NULL
1. Leaf node;
2. Left side exist
3. Right side exist
4. Dono side jaunga;

BST : 1
Size : 2
Min : 10
Max : 20



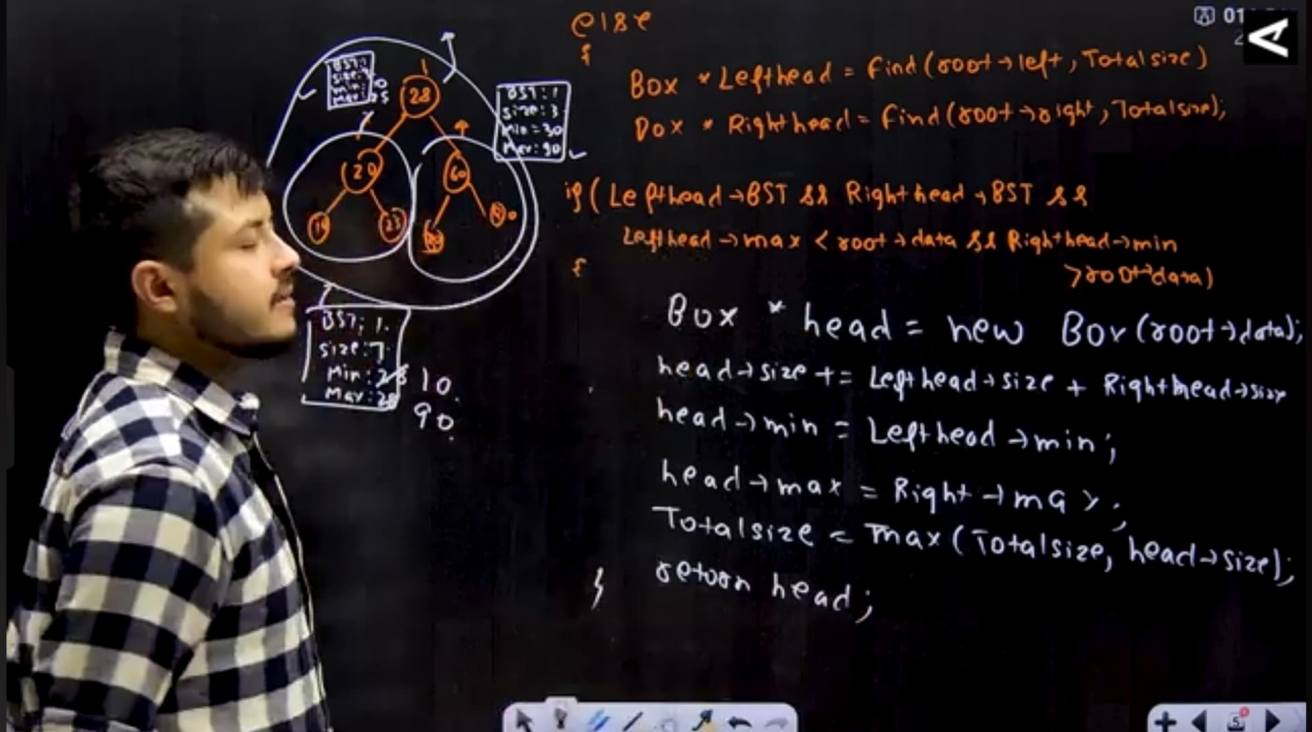
BST : 1
Size : 1
min : 20
max : 20

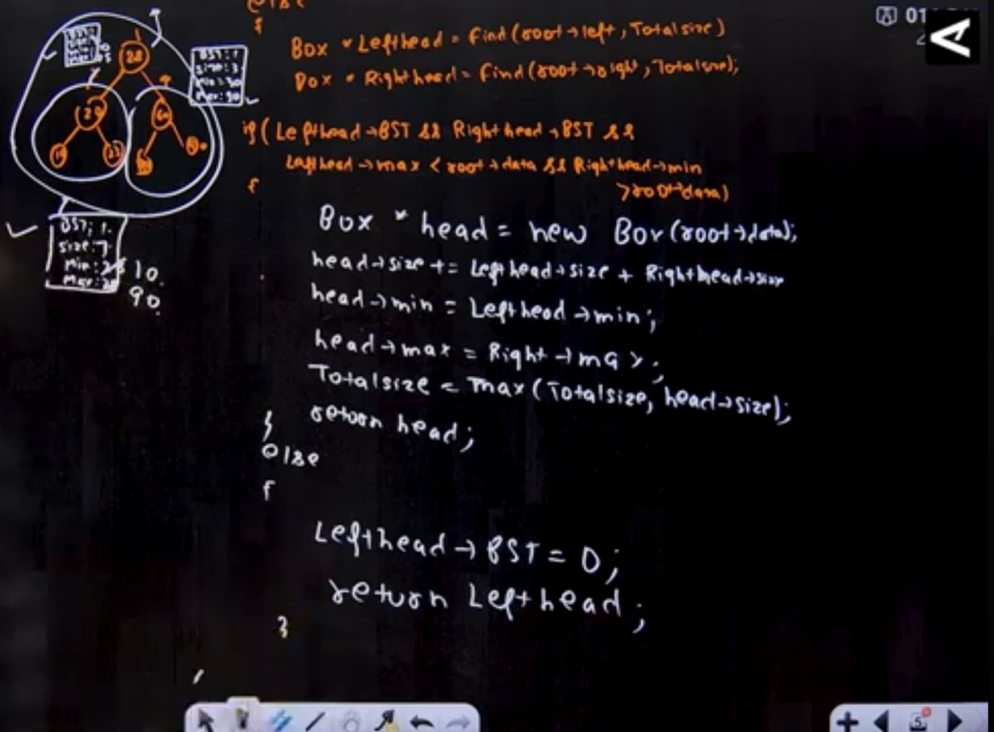
① Leaf node;
② Leftside exist
③ Right side exist
④ Dono side jaunga;


Box* Find (Node *root , int &Totalsize)
{
if (!root->left && !root->right)
return new Box (root->data);
else if (!root->left && root->right)
Box *head = Find(root->right, Totalsize);
if(head->BST && head->min > root->data)
{
head->size++;
head->min = root->data;
Totalsize = max(Totalsize, head->size);
return head;
}
else
{
head->BST = 0;
return head;
}
}
head
BST: 1
size: 24
min: 10
max: 25
10
20
15
25

```
else if ( root->left && !root->right )
{
    Box * head = Find(root->left, Totalsize);

    if ( head->BST && head->max < root->data)
    {
        head->size ++;
        head->max = root->data;
        Totalsize = max(Totalsize, head->size);
        return head;
    }
    else
    {
        head->BST = 0;
        return head;
    }
```


else
{
Box * Lefthead = Find(root->left, Totalsize)
Box * Righthead = Find(root->right, Totalsize);
if(Lefthead->BST && Righthead->BST &&
Lefthead->max < root->data && Righthead->min > root->data)
{
Box * head = new Box(root->data);
head->size += Lefthead->size + Righthead->size
head->min = Lefthead->min;
head->max = Right->max;
Totalsize = max(Totalsize, head->size);
return head;
}
28
20
40
15
23
35
50
BST: 1
size: 3
Min: 30
Max: 90
BST: 1
size: 7
Min: 2
Max: 28
10
90


Box * Lefthead = Find(root->left, Totalsize)
Box * Righthead = Find(root->right, Totalsize);
if(Lefthead->BST && Righthead->BST &&
Lefthead->max < root->data && Righthead->min
>root->data)
Box * head = new Box(root->data);
head->size += Lefthead->size + Righthead->size
head->min = Lefthead->min;
head->max = Right->max;
Totalsize = max(Totalsize, head->size);
return head;
else
Lefthead->BST = 0;
return Lefthead;
}

Problem Editorial Submissions Comments

Largest BST

Medium Accuracy: 29.73% Submissions: 111K+ Points: 4

Given a binary tree. Find the size of its largest subtree that is a Binary Search Tree.

Note: Here Size is equal to the number of nodes in the subtree.

Example 1:

Input:

Output: 1

Explanation: There's no sub-tree with size greater than 1 which forms a BST. All the leaf Nodes are the BSTs with size equal to 1.

Example 2:

C++ (g++ 5.4) Average Time: 40m Start Timer

```
107         int min, max;
108
109         Box(int data)
110         {
111             BST = 1;
112             size =1;
113             min = data;
114             max = data;
115         }
116     };
117
118     Box * find(Node *root, int &Totalsize)
119     {
120         // Leaf Node
121         // Only right side exist
122         // Only left side exist
123         // Both side exist
124     }
125
126
127
128     /*You are required to complete this method */
```

Custom Input Compile & Run Submit

6 8

Output: 1

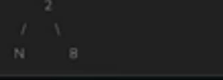
Explanation: There's no sub-tree with size greater than 1 which forms a BST. All the leaf Nodes are the BSTs with size equal to 1.

Example 2:

Input: 6 6 3 N 2 9 3 N 8 8 2

Output: 2

Explanation: The following sub-tree is a BST of size 2:



Your Task:

You don't need to read input or print anything. Your task is to complete the function **largestBst()** that takes the root node of the Binary Tree as its


```
120         // Leaf Node
121         if(!root->left&&!root->right)
122         {
123             return new Box(root->data);
124         }
125         // Only right side exist
126         else if(!root->left&&root->right)
127         {
128             Box *head = find(root->right,Totalsize);
129             // BST yes
130
131             // No
132         }
133         // Only left side exist
134         // Both side exist
135     }
136
137
138
139     /*You are required to complete this method */
140     // Return the size of the largest sub-tree which is
141
```

Problem Editorial Submissions Comments

6 8

Output: 1

Explanation: There's no sub-tree with size greater than 1 which forms a BST. All the leaf Nodes are the BSTs with size equal to 1.

Example 2:

Input: 6 6 3 N 2 9 3 N 8 8 2

Output: 2

Explanation: The following sub-tree is a BST of size 2:

Your Task:

You don't need to read input or print anything. Your task is to complete the function **largestBst()** that takes the root node of the Binary Tree as its

C++ (g++ 5.4) Average Time: 40m Start Timer

```
124         }
125         // Only right side exist
126         else if(!root->left&&root->right)
127         {
128             Box *head = find(root->right,Totalsize);
129             // BST yes
130             if(head->BST&&head->min>root->data)
131             {
132                 head->size++;
133                 head->min = root->data;
134                 Totalsize = max(Totalsize,head->size);
135                 return head;
136             }
137             // No
138             else
139             {
140                 head->BST = 0;
141                 return head;
142             }
143         }
144         // Only left side exist
145
```

Custom Input Compile & Run Submit

Given a binary tree. Find the size of its largest subtree that is a Binary Search Tree.

Note: Here Size is equal to the number of nodes in the subtree.

Example 1:

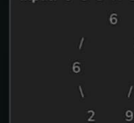
Input:

Output: 1

Explanation: There's no sub-tree with size greater than 1 which forms a BST. All the leaf Nodes are the BSTs with size equal to 1.

Example 2:

Input: 6 6 3 N 2 9 3 N 8 8 2




```
161     // Both side exist
162     else
163     {
164         Box * Lefthead = find(root->left,Totalsize);
165         Box * Righthead = find(root->right,Totalsize);
166
167         if(Lefthead->BST&&Righthead->BST&&Lefthead->max<root
168         {
169             Box *head = new Box(root->data);
170             head->size+= Lefthead->size+Righthead->size;
171             head->min = Lefthead->min;
172             head->max = Righthead->max;
173             Totalsize = max(Totalsize,head->size);
174             return head;
175         }
176         else
177         {
178             Lefthead->BST = 0;
179             return Lefthead;
180         }
181     }
182 }
```

Problem | Editorial | Submissions | Comments

```
   4     4
  /  \
 6    8
```

Output: 1

Explanation: There's no sub-tree with size greater than 1 which forms a BST. All the leaf Nodes are the BSTs with size equal to 1.

Example 2:

Input: 6 6 3 N 2 9 3 N 8 8 2

Output: 2

Explanation: The following sub-tree is a BST of size 2:

Your Task:

C++ (g++ 5.4) | Average Time: 40m | Start Timer

```
110
111         }
112     };
113
114 class Solution{
115     public:
116
117
118     Box * find(Node *root, int &Totalsize)
119     {
120         // root doesn't exist
121         i
122
123         // Root exist
124
125     }
126
127
128
129     /*You are required to complete this method */
130     // Return the size of the largest sub-tree which is
131
```

Custom Input | Compile & Run | Submit

Problem | Editorial | Submissions | Comments

Given a binary tree. Find the size of its largest subtree that is a Binary Search Tree.

Note: Here Size is equal to the number of nodes in the subtree.

Example 1:

Input:

Output: 1

Explanation: There's no sub-tree with size greater than 1 which forms a BST. All the leaf Nodes are the BSTs with size equal to 1.

Example 2:

Input: 6 6 3 N 2 9 3 N 8 8 2

C++ (g++ 5.4) | Average Time: 40m | Start Timer

```
/* Tree node structure  used in the program

struct Node {
    int data;
    Node *left;
    Node *right;

    Node(int val) {
        data = val;
        left = right = NULL;
    }
};*/

                6
1                   1 BST
0                   0 SIze
min =. INT_MAX.     min =. INT_MAX
max = INT_MIN       max = INT_MIN

 class Box{
    public:
```

Custom Input | Compile & Run | Submit

Problem Editorial Submissions Comments

Given a binary tree. Find the size of its largest subtree that is a Binary Search Tree.

Note: Here Size is equal to the number of nodes in the subtree.

Example 1:

Input:

Output: 1

Explanation: There's no sub-tree with size greater than 1 which forms a BST. All the leaf Nodes are the BSTs with size equal to 1.

Example 2:

Input: 6 6 3 N 2 9 3 N 8 8 2

C++ (g++ 5.4) Average Time: 40m Start Timer

```
99      }
100
101
102 };*/
103
104 1
105 1
106 6
107 6
108                    6
109 1                  1 BST
110 0                  0 SIze
111 min =. INT_MAX.    min =. INT_MAX
112 max = INT_MIN      max = INT_MIN
113
114  class Box{
115      public:
116      bool BST;
117      int size;
118      int min, max;
119
```

Custom Input Compile & Run Submit

Maximum Sum BST in Binary Tree