Day - 47

(BST - Hard problems)

problem 1 : preorder to postorder

~~Po~~ pre : 40 30 35 80 100

In BST, the left subtree is smaller than root and right subtree is bigger than root

Start creation,

40 →

root

40

30 →

30 < 40

left side

40
/
30

35 →

35 > 30

35 < 40

right of 35

40
/
30
\
35

80 →

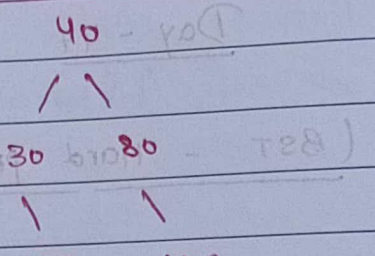
80 > 40

right of 40

40
/ \
30 80
\
35

100

100 > 80



We approach this problem as range.

MIN = INT_MIN

MAX = INT_MAX

The first Number from preorder:

from start,

If any Node want to Make BST.

MIN < Element < MAX

40 →

(40)

Divided in 2 parts.

(40)

Min < Element < 40

40 < Element < MAX

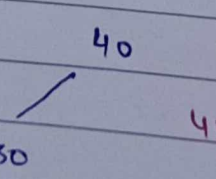
30 →

check on left

30 > MIN ✓

30 < 40 ✓

push on left



40 < Element < MAX

MIN < Element < 30

30 < Element < 40

35 →

check on left

35 > MIN ✓

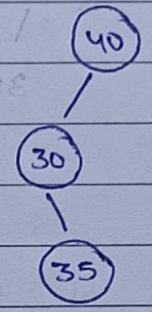
35 < 30 ✗ fails

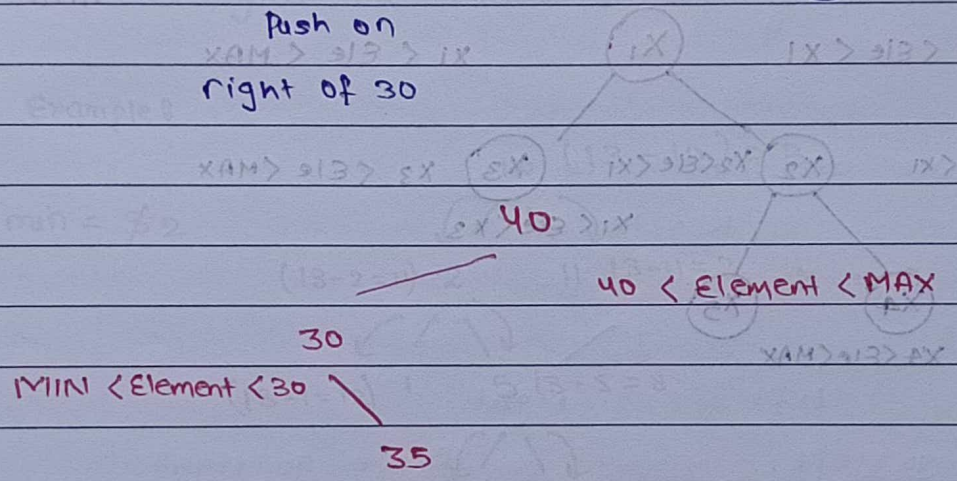
return, check on right

35 > 30 ✓

35 < 40 ✓

Push on right of 30



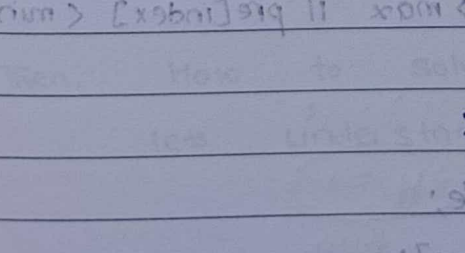


80 →

check left of 35 : 80 > 30 ✓ 80 < 35 ✗

check right of 35 : 80 > 35 ✓ 80 < 40 ✗

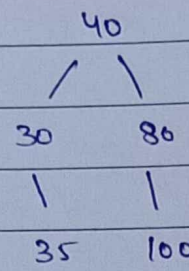
return check right of 40 : 80 > 40 ✓ 80 < MAX ✓



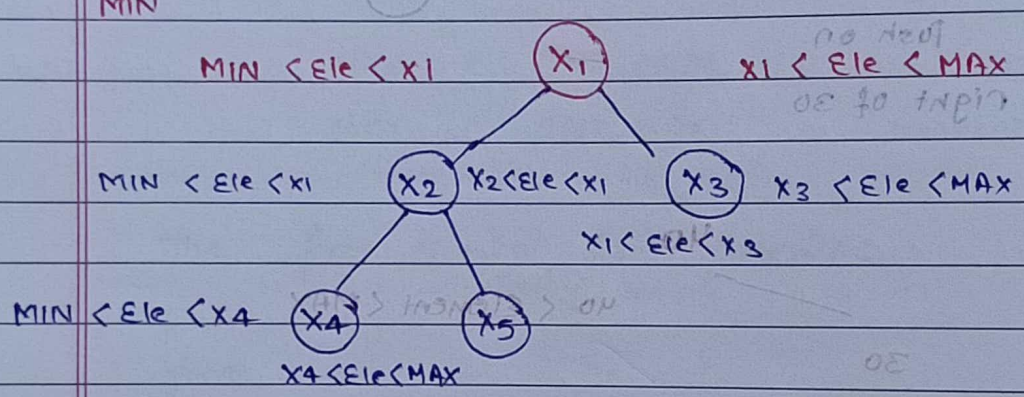
100 :

Check left of 80 : 100 > 40 ✓ 100 < 80 ✗

Check right of 80 : 100 > 80 ✓ 100 < MAX ✓



Required ans.



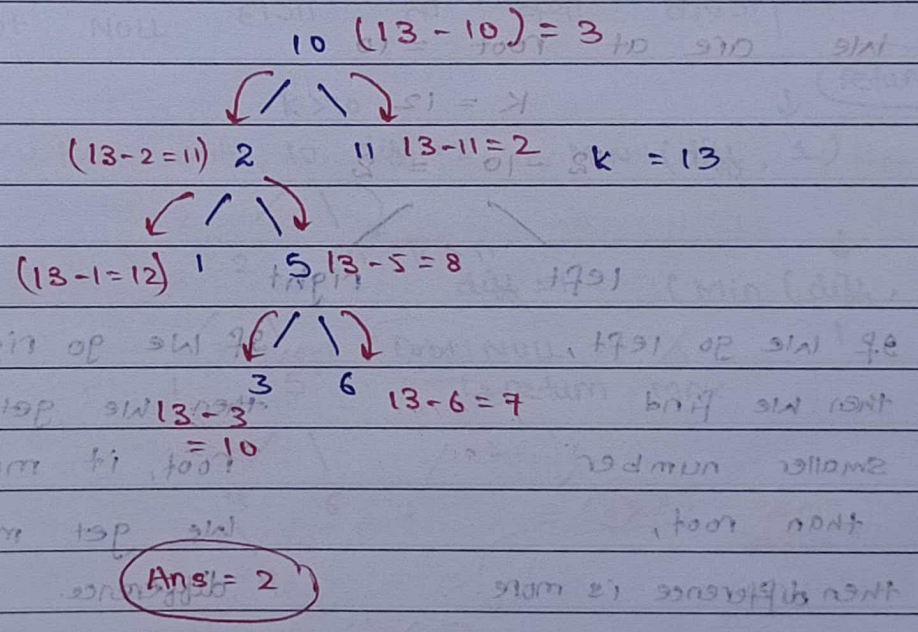
Code :

```
Node* post_order (int pre[], int size) {

    int min = INT_MIN, max = INT_MAX, index = 0;
    return find(pre, min, max, index, size);
}

Node * find (int pre[], int min, int max, int &index, int size){
        if (index >= size || pre[index] > max || pre[index] < min)
            return NULL;

        Node * root = new Node;
            root -> data = pre [index];
            index++;
            if (index == size)
            return root;
```

```
root -> left = find (pre, min, root -> data, index, size);

root -> right = find (pre, root -> data, max, index, size);

return root;
}
```

Problem 2: find the Closest Element in BST :

Example:

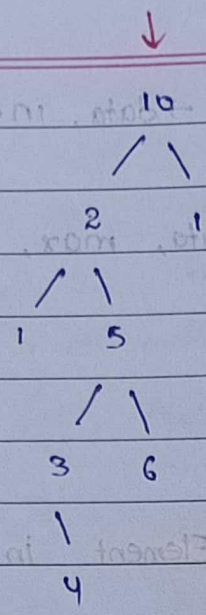


is it necessary to check all Elements

No.

Then, How to solve this

lets understand with Example.



We are at root = 10

K = 12

12 - 10 = 2

left

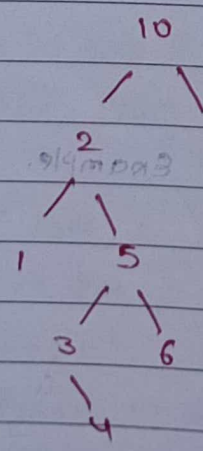
right

If we go left, then we find smaller number than root, then difference is more.

If we go right then we get bigger than root, it may possible we get more close difference.

So, we move right.

diff = 2



(12 - 11) = 1

diff = 1

min

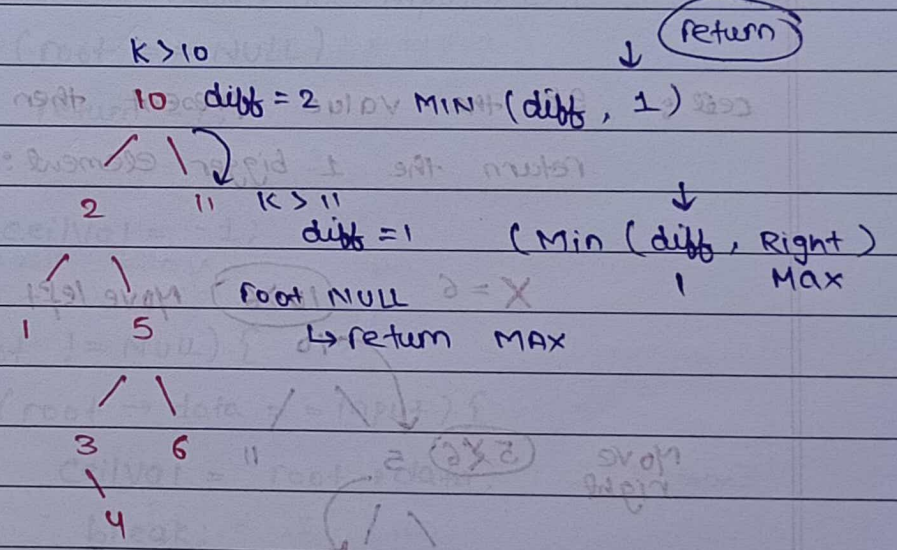
11 is leaf then return.

case 1: K = root → data
return 0;

case 2: K > root → data
↳ go to right side

case 3: K < root → data
↳ go to left side.

अगर root NULL होगा तो क्या करेंगे।

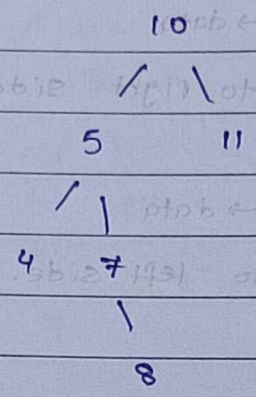


code:

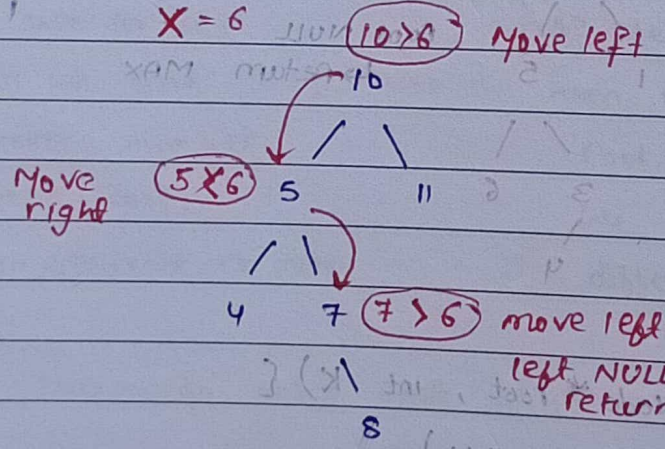
```
int minDiff (Node * root, int K) {
    if (root == NULL)
        return INT_MAX;

    if (K == root→data)
        return 0;

    if (K > root→data)
    return min(K - root→data, minDiff (root→right, K));

    if (K < root→data)
    return min(root→data - K, minDiff (root→left, K));
}
```

Problem 3: Ceil in BST :

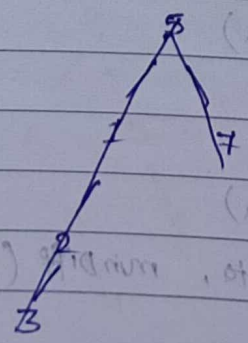
Example 1



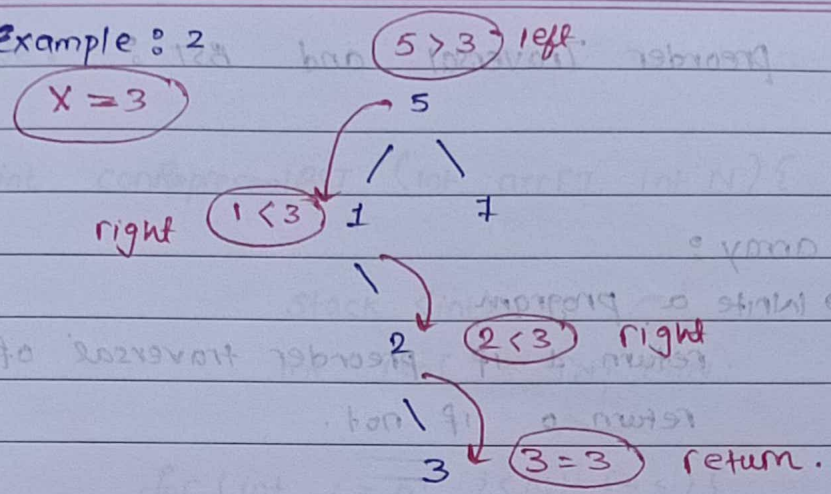
ceil : If the value present then return, otherwise return the 1 bigger element :



Example 2 :



Example : 2



Code :

```
int findceil (Node * root, int input){
        if (root == NULL)
            return -1;

    int ceilval = -1;

    While (root != NULL) {
        if (root → data == input) {
            ceilval = root → data;
            break;
        } else if (root → data < input) {
            root = root → right;
        } else {
            ceilval = root → data;
            root = root → left;
        }
    }
    return ceilval;
}
```

problem 4: preorder Traversal and BST:

Given an array:

↳ Write a program

return 1 if preorder traversal of BST.

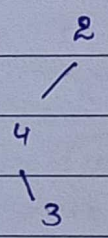
return 0 if not.

Example 1:

N = 3

arr = {2, 4, 3}

output = 1

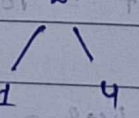


valid preorder.

Example: 2

arr = {2, 4, 1}

NLR



preorder: [2, 1, 4]

arr is not preorder

↳ return 0.

Code :

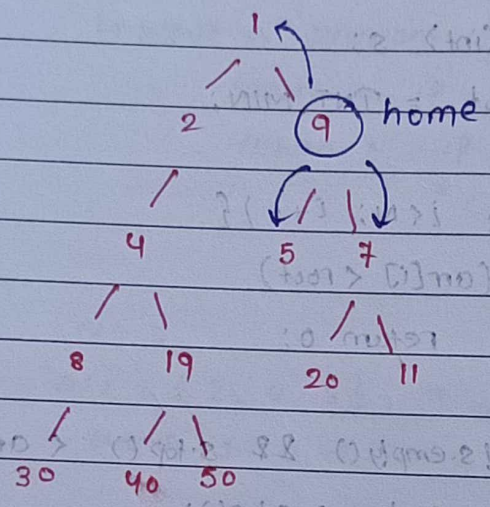
```
int canRepresentBST (int arr[], int N){

        stack <int> s;
        int root = INT_MIN;

    for (int i = 0; i<N; i++){
            if (arr[i] < root)
                return 0;

            while ( !s.empty() && s.top() < arr[i]){
                root = s.top();
                s.pop();
            }

            s.push (arr[i]);
        }
        return 1;
}
```

problem 5 : Bheem wants Ladoos :

Example 1

K = 1

distance.



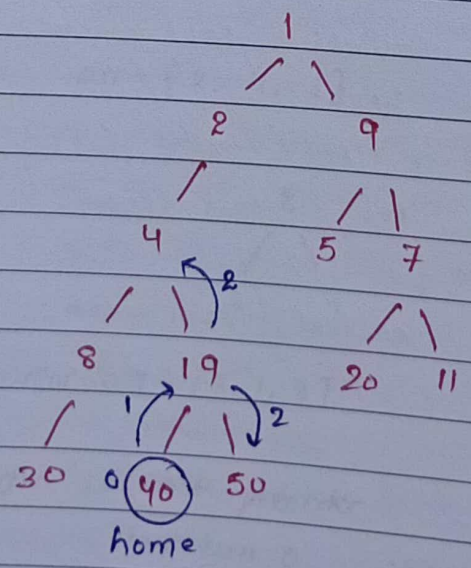
9 + 1 + 5 + 7 = 22

output.

Example 2

K = 2

distance.



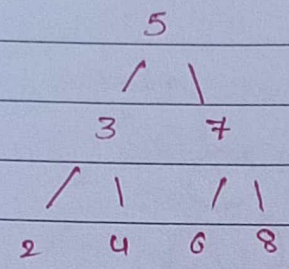
40 + 19 + 4 + 50

= 113

output.

problem 6:

MINIMUM ABSOLUTE DIFFERENCE in BST



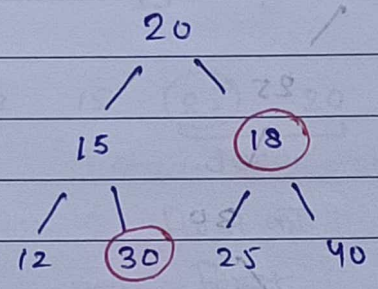
code:

```
void inorder(Node* root, int &prev, int &minDiff){
        if (root == NULL)
            return;

    inorder(root->left, prev, minDiff);

        if (prev != -1)
            minDiff = min(minDiff, root->data - prev);
    prev = root->data;

    inorder(root->right, prev, minDiff);
}

int absolute_diff (Node* root){
        int prev = -1;
        int minDiff = INT_MAX;
        inorder(root, prev, minDiff);
    return minDiff;
}
```

Problem 7: Fixing two swapped Nodes of a BST:

Example

BST hai lekin 2 node Galat Jagah pe hai
wo swapped ऐ Rakha है।
usi ko find karna है।



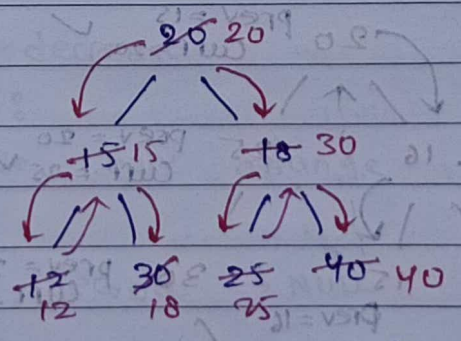
Inorder: 12 15 30 20 25 18 40
inc inc dec inc dec

Swap(30, 18)

In inorder it always increase.

After swapping:

12 15 18 20 25 30 40.

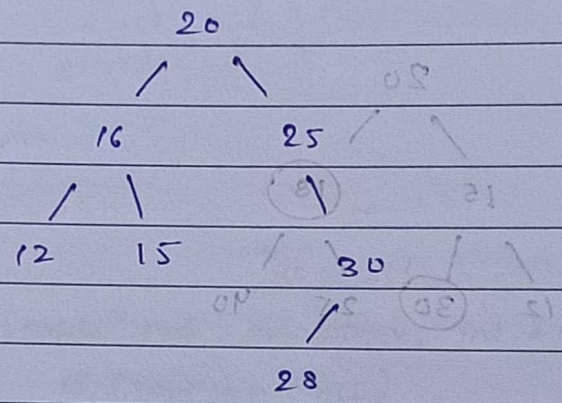


घुमते जाओ Put करते जाओ।

→ Now we check the edge cases :

अगर सिर्फ एक बार Array में decrease हो रहा है तो :-

Example :



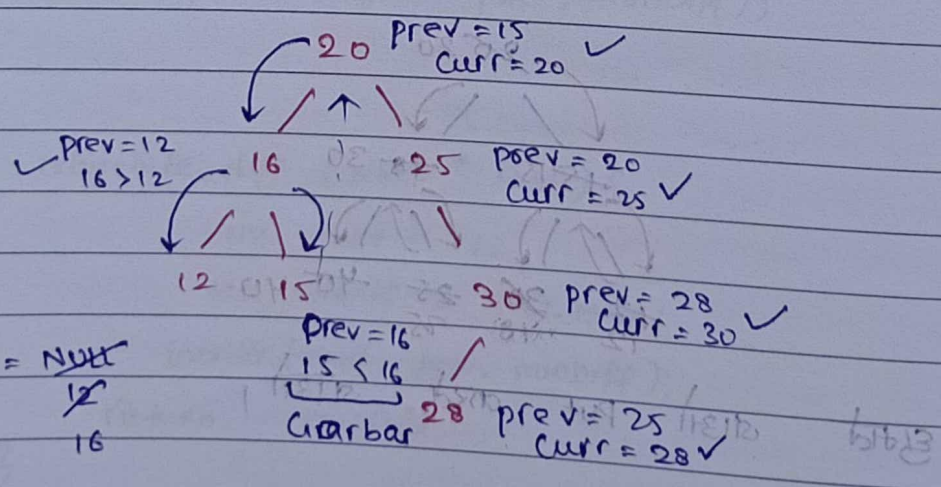
Inorder : 12 16 15 20 25 28 30

inc dec inc inc inc inc

If dec at only one place

then swap the both element.

Without creating array, how we solve :

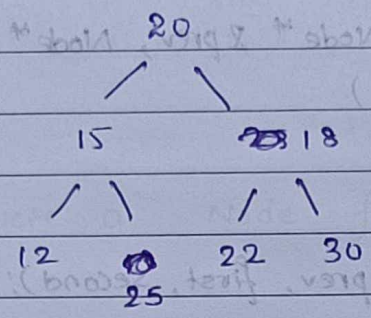


first = prev

second = root.

then swap this.

If two decrease :



Inorder : 12 15 25 20 22 18 30

dec dec

first का second का

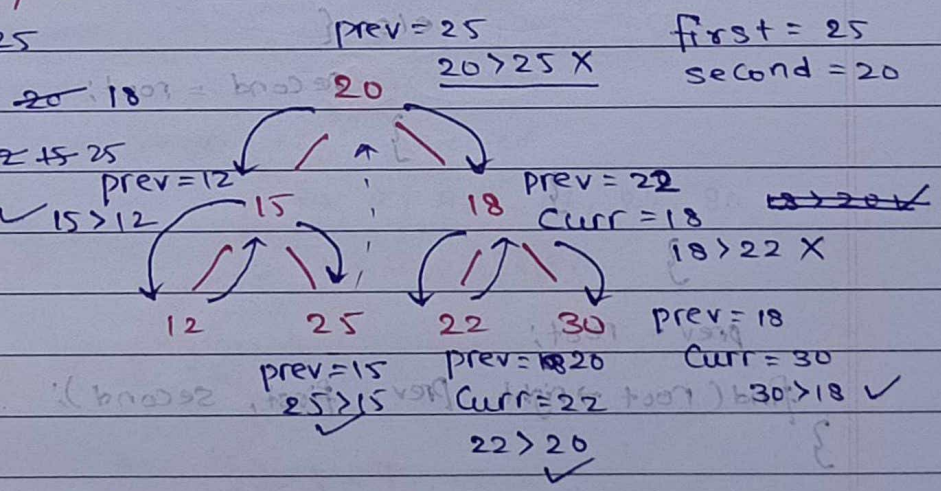
first second.

Without array :

First = ~~NULL~~ 25

second = ~~NULL~~ ~~20~~ 18

prev = ~~NULL~~ ~~12~~ ~~15~~ 25



When we found decrease 2nd time :

We only change end pointer. (second).

① prev, first, second → NULL

② Inorder traversal

prev exist

→ value decrease

↳ 1st time if(first = NULL & second = NULL)

↳ 2nd time if(first != NULL).

Code :

```
Void find (Node* root, Node* &prev, Node* &first, Node* &second){
        if (!root)
            return;

    find (root->left, prev, first, second);

    if (prev) {
        if (prev->data>root->data){
            if (!first && !second){
                first = prev;
                second = root;
            }
            else{
                second = root;
            }
        }
    }
    prev = root;
    find(root->right, prev, first, second);
}

struct Node * correctBST (struct Node* root){
    Node* prev = NULL, *first = NULL, *second = NULL;
    find (root, prev, first, second);
    int data = first->data;
    first->data = second->data;
    second->data = data;
    return root;
}
```