**Floor in BST: -**

**Medium** Accuracy: **51.06%** Submissions: **33K+** Points: **4**

You are given a BST(Binary Search Tree) with **n** number of nodes and value **x**. your task is to find the greatest value node of the BST which is smaller than or equal to x.  
**Note:** when x is smaller than the smallest node of BST then returns -1.

**Example:**

**Input:**

n = 7 2

\

81

/ \

42 87

\ \

66 90

/

45

x = 87

**Output:**

87

**Explanation:**

87 is present in tree so floor will be 87.

**Example 2:**

**Input:**

n = 4 6

\

8

/ \

7 9

x = 11

**Output:**

9

**Your Task:-**  
You don't need to read input or print anything. Complete the function **floor()**which takesthe integer **n** and BST and integer x returns the floor value.

**Constraint:**  
1 <= Noda data <= 109  
1 <= n <= 105

**Expected Time Complexity:** O(n)  
**Expected Space Complexity:** O(1)

**Code: -**

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

struct Node {

int data;

Node \*right;

Node \*left;

Node(int x) {

data = x;

right = NULL;

left = NULL;

}

};

// } Driver Code Ends

// Function to search a node in BST.

class Solution{

public:

int floor(Node\* root, int x) {

// Code here

int ans = -1;

while(root){

if(x == root->data)

return root->data;

else if(x < root->data)

root = root->left;

else{

ans = root->data;

root = root->right;

}

}

return ans;

}

};

//{ Driver Code Starts.

Node \*insert(Node \*tree, int val) {

Node \*temp = NULL;

if (tree == NULL) return new Node(val);

if (val < tree->data) {

tree->left = insert(tree->left, val);

} else if (val > tree->data) {

tree->right = insert(tree->right, val);

}

return tree;

}

int main() {

int T;

cin >> T;

while (T--) {

Node \*root = NULL;

int N;

cin >> N;

for (int i = 0; i < N; i++) {

int k;

cin >> k;

root = insert(root, k);

}

int s;

cin >> s;

Solution obj;

cout << obj.floor(root, s) << "\n";

}

}

// } Driver Code Ends

**T.C: - O(N), N = height of BST**

**S.C: - O(1)**