Floor in Binary Search Tree (BST)

Given a Binary Search Tree and a number x, find the floor of x in the given BST:

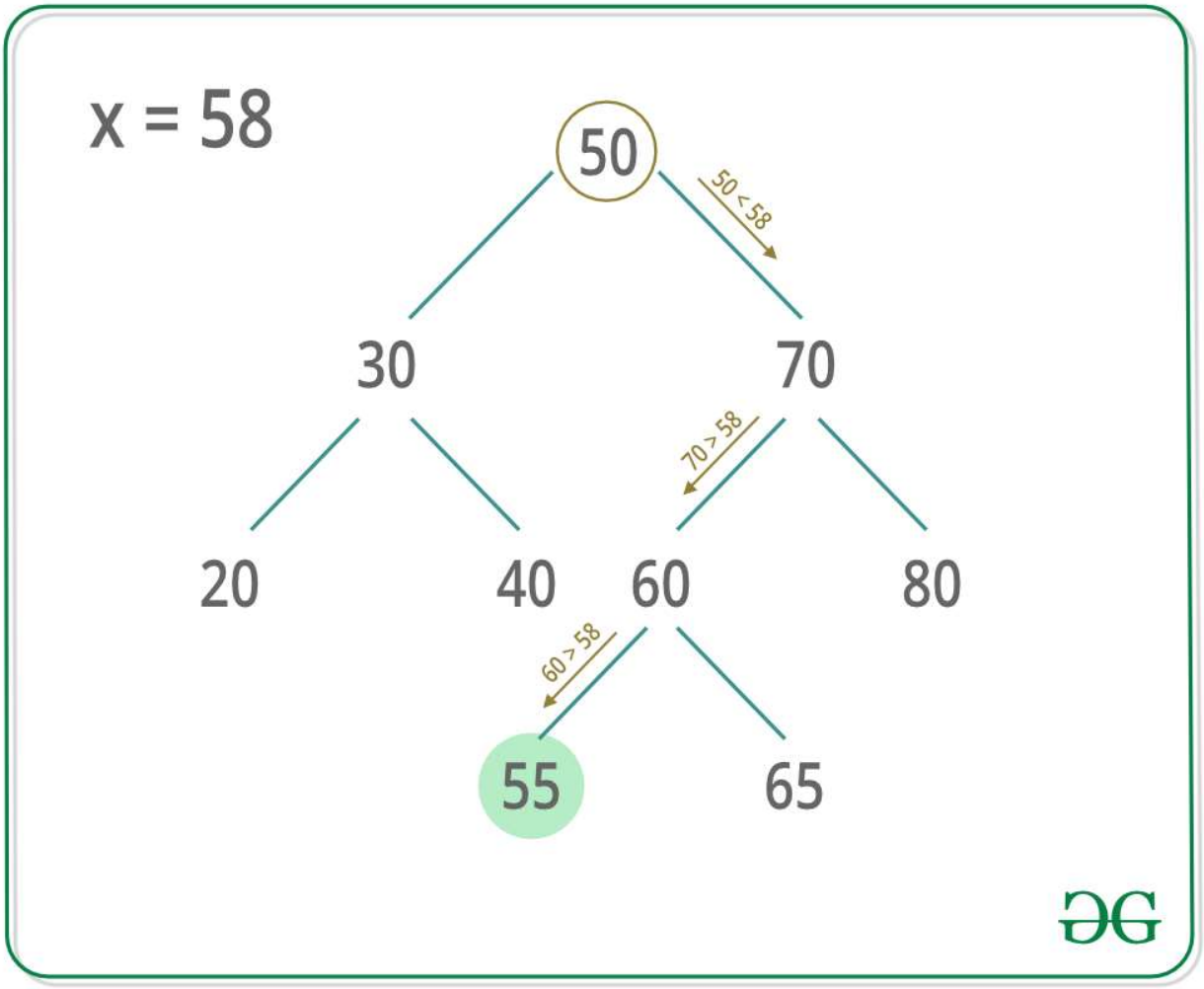
Examples:

Input: *x = 14 and root of below tree*

Output: *12*

Input: *x = 15 and root of below tree*

Output: *15*



Naive Approach: To solve the problem follow the below idea:
*A **simple solution** is to traverse the tree using (Inorder or Preorder or Postorder) and keep track of the closest smaller or same element*
Efficient Approach: To solve the problem follow the below idea:
*We can **efficiently** find the closest smaller or same element in **O(H)** time where H is the height of BST.*
Follow the given steps to solve the problem:

- Start at the root Node
- If root->data == key, the floor of the key is equal to the root.
- Else if root->data > key, then the floor of the key must lie in the left subtree.
- Else floor may lie in the right subtree but only if there is a value lesser than or equal to the key.
- If not, then the root is the key.

Below is the implementation of the above approach:

C++

```
// C++ code to find floor of a key in BST
#include <bits/stdc++.h>
using namespace std;

/*Structure of each Node in the tree*/
struct Node {
    int data;
    Node *left, *right;
};

/*This function is used to create and
initializes new Nodes*/
Node* newNode(int key)
{
    Node* temp = new Node;
    temp->left = temp->right = NULL;
    temp->data = key;
    return temp;
}

/* This function is used to insert
 new values in BST*/
Node* insert(Node* root, int key)
{
    if (!root)
```

```
        return newNode(key);
    if (key < root->data)
        root->left = insert(root->left, key);
    else
        root->right = insert(root->right, key);
    return root;
}

/*This function is used to find floor of a key*/
int floor(Node* root, int key)
{
    if (!root)
        return INT_MAX;

    /* If root->data is equal to key */
    if (root->data == key)
        return root->data;

    /* If root->data is greater than the key */
    if (root->data > key)
        return floor(root->left, key);

    /* Else, the floor may lie in right subtree
      or may be equal to the root*/
    int floorValue = floor(root->right, key);
    return (floorValue <= key) ? floorValue : root->data;
}
```

```
// Driver code
int main()
{
    /* Let us create following BST
              7
            /    \
           5      10
         /  \    /  \
        3    6   8   12 */
    Node* root = NULL;
    root = insert(root, 7);
    insert(root, 10);
    insert(root, 5);
    insert(root, 3);
    insert(root, 6);
    insert(root, 8);
    insert(root, 12);

      // Function call
    cout << floor(root, 9) << endl;
    return 0;
}
```

Output
8

Time Complexity: O(H), where H is the height of the tree
Auxiliary Space: O(1)
