15) MEDIAN OF BST IN O(N) TIME AND O(1) SPACE.

METHOD 1:
In this method, we just count the number of nodes by InOrder traversal and during this traversal we simultaneously convert the BST into Doubly Linked List(DLL).
Then we use the slow and fast pointer approach to reach the mid of the linked list.
Then the required output is given depending weather the count is even or odd.
(This method uses the recursive stack space of O(N))

METHOD 2:
This is very special method knows **MORRIS TRAVERSAL**
LINK: **L37. Morris Traversal | Preorder | Inorder | C++ | Java**

CODE OF METHOD 1:

```
void InOrderTraversal(Node *root,int &count,Node *&previous,Node *&start){
   if(root!=NULL){
      InOrderTraversal(root->left,count,previous,start);
      count++;
      if(previous==NULL){
         root->left=previous;
         previous=root;
         start=root;
      }
      else{
         root->left=previous;
         previous->right=root;
         previous=root;
      }
      InOrderTraversal(root->right,count,previous,start);
   }
}

float findMedian(struct Node *root)
{
   int count=0;
   Node *previous=NULL;
   Node *start=NULL;
   InOrderTraversal(root,count,previous,start);
   Node *slow=start;
   Node *fast=start;
   while(fast->right!=NULL && fast->right->right!=NULL){
      slow=slow->right;
      fast=fast->right->right;
   }
   if(count%2==0){
      return float(float(slow->data+slow->right->data)/2);
   }
   else{
```

```
        return float(slow->data);
    }
}
```

CODE OF METHOD 2:

FUNCTION FOR COUNTING THE NUMBER OF NODES IN THE BST:

```
int countFunction(Node *root,Node *&start){
    int count=0;
    Node *current=root;
    while(current!=NULL){
        if(current->left==NULL){
            count++;
            if(count==1){
                start=current;
            }
            current=current->right;
        }
        else{
            Node *t=current->left;
            while(t->right!=NULL && t->right!=current){
                t=t->right;
            }
            if(t->right==NULL){
                t->right=current;
                current=current->left;
            }
            else{
                //t->right=NULL;
                count++;
                if(count==1){
                    start=current;
                }
                current=current->right;
            }
        }
    }
    return count;
}
```