Question - 1

Creating a Binary Search Tree

Consider the algorithm below which takes an array, *keys*, of *n* unique integers and inserts each integer *in order* into an empty binary search tree:

```
// Before: create a variable named 'counter' that counts
calls to insert. Initialize 'counter' to 0.

createBST(int[] keys) {
    set the value of counter to 0

    for each (key in keys) {
        if (tree has a root node) {
            insert(root, key)
        }
        else {
            create a new node with value 'key' as the root node
of tree
        }

        print the value of 'counter' on a new line
    }
}

insert(root, key) {
    increment the value of counter by 1

    if (key is less than the value of root node) {
        if (root node has no left child) {
            create a new node with value 'key' as the left child
of root node
        }
        else {
            insert(left child of root node, key)
        }
    }
    else {
        if (root node has no right child) {
            create a new node with value 'key' as the right child
of root node
        }
        else {
            insert(right child of root node, key)
        }
    }
}
```

Note: Recall that a Binary Search Tree is a Binary Tree in which all elements in a node's left subtree are $\leq$ all elements in the node's right subtree.

Complete the *createBST* function in the editor below. It has *1* parameter: an array of *n* unique integers, *keys*. The function must implement the above algorithm as well as any additional variables, functions, or classes necessary to make the implementation work.

Input Format

Locked stub code in the editor reads the following input from stdin and passes it to the function:

the *keys* array.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer describing the value of element *i* in *keys*.

Constraints

- $1 \le n \le 2 \times 10^5$
- $1 \le keys_i \le n$
- The *keys* array contains no duplicate elements.

Output Format

The function must print the value of the *counter* variable on a new line after each insertion of a key into the tree.

Sample Input 0

```
3
2
1
3
```

Sample Output 0

```
0
1
2
```

Explanation 0

We perform the following sequence of insertions:

1. $key = 2$: As the tree is currently empty, we create a new node with value *2* and designate it as the root node. We then print the value of *counter*; as no calls to the *insert* function have been made yet, we print *0* on a new line.
2. $key = 1$: As the tree has a root node, we make a call to the *insert* function. The *insert* function only runs once before the node is inserted into the root's left subtree, so the value of *counter* is only incremented once. We then print the value of the counter, *1*, on a new line.
3. $key = 3$: As the tree has a root node, we make a call to the *insert* function. The *insert* function only runs once before the node is inserted into the root's right subtree, so the value of *counter* is only incremented once; as *counter* was already incremented once during the previous insertion, its value is now *2*. We then print the value of *counter*, *2*, on a new line.

Sample Input 1

```
3
1
2
3
```

Sample Output 1

```
0
1
3
```

Explanation 1

We perform the following sequence of insertions:

1. $key = 1$: As the tree is currently empty, we create a

new node with value *1* and designate it as the root node. We then print the value of *counter*; as no calls to the *insert* function have been made yet, we print *0* on a new line.

2. *key = 2*: As the tree has a root node, we make a call to the *insert* function. The *insert* function only runs once before the node is inserted into the root's right subtree, so the value of *counter* is only incremented once. We then print the value of the counter, *1*, on a new line.
3. *key = 3*: As the tree has a root node, we make a call to the *insert* function. The *insert* function runs twice before the node is inserted into the right subtree of the root's right subtree, so the value of *counter* is incremented twice; as *counter* was already incremented once during the previous insertion, its value is now *3*. We then print the value of *counter*, *3*, on a new line.