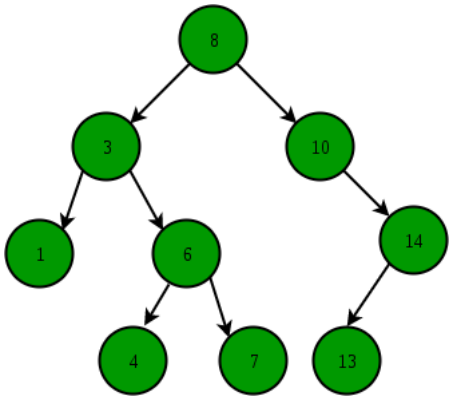
Binary Search Tree | Set 1 (Search and Insertion)

The following is definition of Binary Search Tree(BST) according to Wikipedia

Binary Search Tree, is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.
  There must be no duplicate nodes.



The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search a given key.

Searching a key

To search a given key in Binary Search Tree, we first compare it with root, if the key is present at root, we return root. If key is greater than root's key, we recur for right subtree of root node. Otherwise we recur for left subtree.

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

C/C++

```
// C function to search a given key in a given BST
struct node* search(struct node* root, int key)
{
    // Base Cases: root is null or key is present at root
    if (root == NULL || root->key == key)
       return root;

    // Key is greater than root's key
    if (root->key < key)
       return search(root->right, key);
```

```
    // Key is smaller than root's key
    return search(root->left, key);
}
```

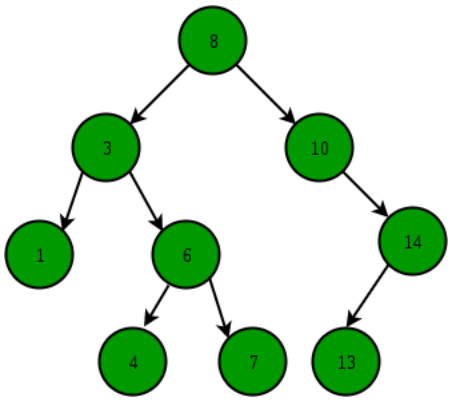
Python

```
# A utility function to search a given key in BST
def search(root,key):

    # Base Cases: root is null or key is present at root
    if root is None or root.val == key:
        return root

    # Key is greater than root's key
    if root.val < key:
        return search(root.right,key)

    # Key is smaller than root's key
    return search(root.left,key)

# This code is contributed by Bhavya Jain
```

Java

```
// A utility function to search a given key in BST
public Node search(Node root, int key)
{
    // Base Cases: root is null or key is present at root
    if (root==null || root.key==key)
        return root;

    // val is greater than root's key
    if (root.key > key)
        return search(root.left, key);

    // val is less than root's key
    return search(root.right, key);
}
```

Illustration to search 6 in below tree:
1. Start from root.
2. Compare the inserting element with root, if less than root, then recurse for left, else recurse for right.
3. If element to search is found anywhere, return true, else return false.



Insertion of a key

A new key is always inserted at leaf. We start searching a key from root till we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.

```
          100                               100
         /   \        Insert 40            /    \
       20     500    --------->          20     500
      /  \                               /  \
```

```
     10   30                          10   30
                                            \
                                            40
```

C/C++

```
// C program to demonstrate insert operation in binary search tree
#include<stdio.h>
#include<stdlib.h>

struct node
{
    int key;
    struct node *left, *right;
};

// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp =  (struct node *)malloc(sizeof(struct node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

// A utility function to do inorder traversal of BST
void inorder(struct node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d \n", root->key);
        inorder(root->right);
    }
}

/* A utility function to insert a new node with given key in BST */
struct node* insert(struct node* node, int key)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    /* return the (unchanged) node pointer */
    return node;
}

// Driver Program to test above functions
int main()
{
    /* Let us create following BST
              50
           /     \
          30      70
         /  \    /  \
       20   40  60   80 */
    struct node *root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    // print inoder traversal of the BST
    inorder(root);

    return 0;
}
```

Python

```
# Python program to demonstrate insert operation in binary search tree

# A utility class that represents an individual node in a BST
class Node:
    def __init__(self,key):
        self.left = None
        self.right = None
        self.val = key

# A utility function to insert a new node with the given key
def insert(root,node):
    if root is None:
        root = node
    else:
        if root.val < node.val:
            if root.right is None:
                root.right = node
            else:
                insert(root.right, node)
        else:
            if root.left is None:
                root.left = node
            else:
                insert(root.left, node)

# A utility function to do inorder tree traversal
def inorder(root):
    if root:
        inorder(root.left)
        print(root.val)
        inorder(root.right)


# Driver program to test the above functions
# Let us create the following BST
#      50
#    /      \
#   30     70
#   / \    / \
#  20 40  60 80
r = Node(50)
insert(r,Node(30))
insert(r,Node(20))
insert(r,Node(40))
insert(r,Node(70))
insert(r,Node(60))
insert(r,Node(80))

# Print inoder traversal of the BST
inorder(r)

# This code is contributed by Bhavya Jain
```

Java

```
// Java program to demonstrate insert operation in binary search tree
class BinarySearchTree {

    /* Class containing left and right child of current node and key value*/
    class Node {
        int key;
        Node left, right;

        public Node(int item) {
            key = item;
            left = right = null;
        }
    }

    // Root of BST
    Node root;

    // Constructor
    BinarySearchTree() {
        root = null;
```

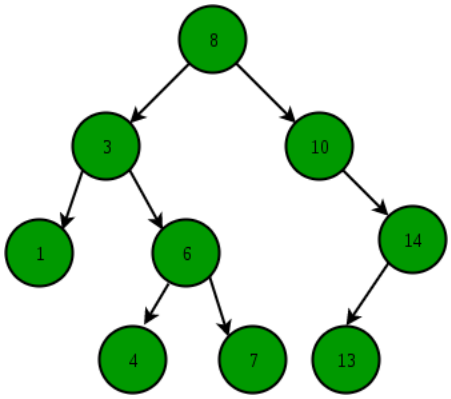
```
    }

    // This method mainly calls insertRec()
    void insert(int key) {
       root = insertRec(root, key);
    }

    /* A recursive function to insert a new key in BST */
    Node insertRec(Node root, int key) {

        /* If the tree is empty, return a new node */
        if (root == null) {
            root = new Node(key);
            return root;
        }

        /* Otherwise, recur down the tree */
        if (key < root.key)
            root.left = insertRec(root.left, key);
        else if (key > root.key)
            root.right = insertRec(root.right, key);

        /* return the (unchanged) node pointer */
        return root;
    }

    // This method mainly calls InorderRec()
    void inorder()  {
       inorderRec(root);
    }

    // A utility function to do inorder traversal of BST
    void inorderRec(Node root) {
        if (root != null) {
            inorderRec(root.left);
            System.out.println(root.key);
            inorderRec(root.right);
        }
    }

    // Driver Program to test above functions
    public static void main(String[] args) {
        BinarySearchTree tree = new BinarySearchTree();

        /* Let us create following BST
              50
           /     \
          30      70
         /  \    /  \
        20   40  60   80 */
        tree.insert(50);
        tree.insert(30);
        tree.insert(20);
        tree.insert(40);
        tree.insert(70);
        tree.insert(60);
        tree.insert(80);

        // print inorder traversal of the BST
        tree.inorder();
    }
}
// This code is contributed by Ankur Narain Verma
```

Output:

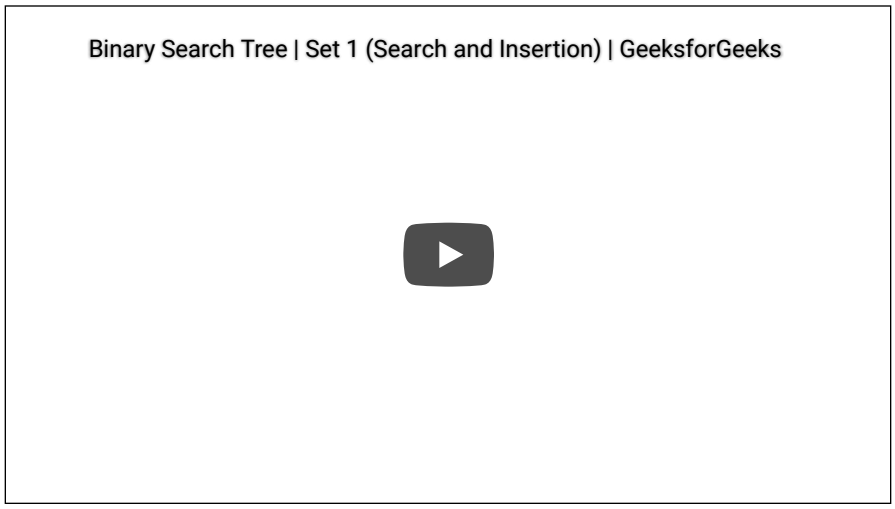
```
20
30
40
50
60
70
80
```

Illustration to insert 2 in below tree:

1. Start from root.
2. Compare the inserting element with root, if less than root, then recurse for left, else recurse for right.
3. After reaching end,just insert that node at left(if less than current) else right.



Time Complexity: The worst case time complexity of search and insert operations is O(h) where h is height of Binary Search Tree. In worst case, we may have to travel from root to the deepest leaf node. The height of a skewed tree may become n and the time complexity of search and insert operation may become O(n).



Some Interesting Facts:

- Inorder traversal of BST always produces sorted output.
- We can construct a BST with only Preorder or Postorder or Level Order traversal. Note that we can always get inorder traversal by sorting the only given traversal.
- Number of unique BSTs with n distinct keys is Catalan Number

Related Links:

- Binary Search Tree Delete Operation
- Quiz on Binary Search Tree
- Coding practice on BST
- All Articles on BST

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Recommended Posts:

Count the Number of Binary Search Trees present in a Binary Tree

Make Binary Search Tree

Optimal Binary Search Tree | DP-24

Binary Search Tree | Set 2 (Delete)

Floor in Binary Search Tree (BST)

Sum of all the levels in a Binary Search Tree

How to handle duplicates in Binary Search Tree?

Print all even nodes of Binary Search Tree

Print Binary Search Tree in Min Max Fashion

Iterative searching in Binary Search Tree

Construct a Binary Search Tree from given postorder

Number of pairs with a given sum in a Binary Search Tree

Threaded Binary Search Tree | Deletion

Inorder Successor in Binary Search Tree

Print all odd nodes of Binary Search Tree

Article Tags : Binary Search Tree | Amazon | Linkedin | Microsoft | Samsung

Practice Tags : Amazon | Microsoft | Samsung | Linkedin | Binary Search Tree

32

To-do Done

1.8

Based on **169** vote(s)

Feedback/ Suggest Improvement | Notes | Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments