Search given key in binary search tree

Given a **BST**, the task is to search a node in this **BST**.
For searching a value in BST, consider it as a sorted array. Now we can easily perform search operation in BST using ***Binary Search Algorithm***.

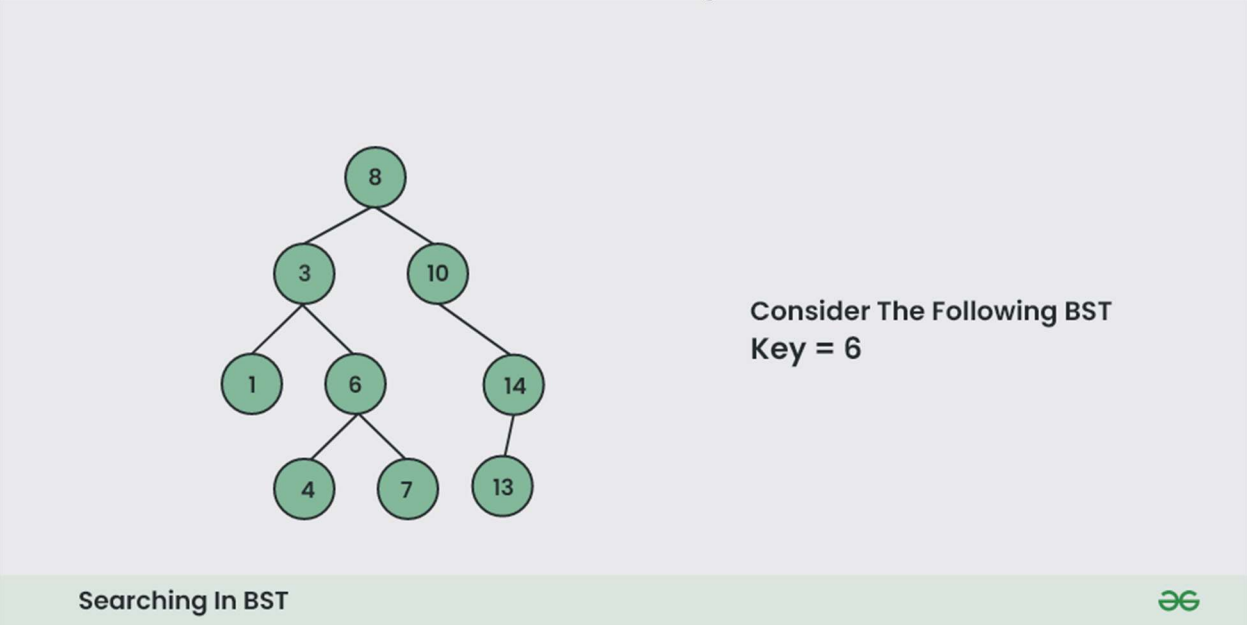
Algorithm to search for a key in a given Binary Search Tree:

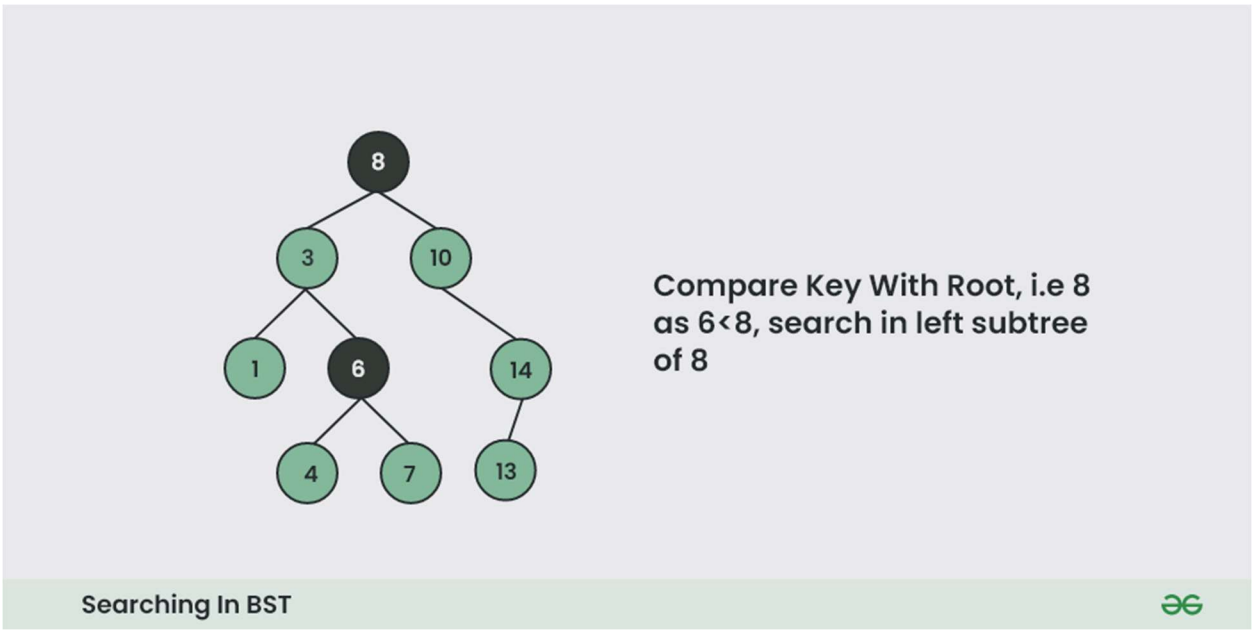
Let's say we want to search for the number **X,** We start at the root. Then:

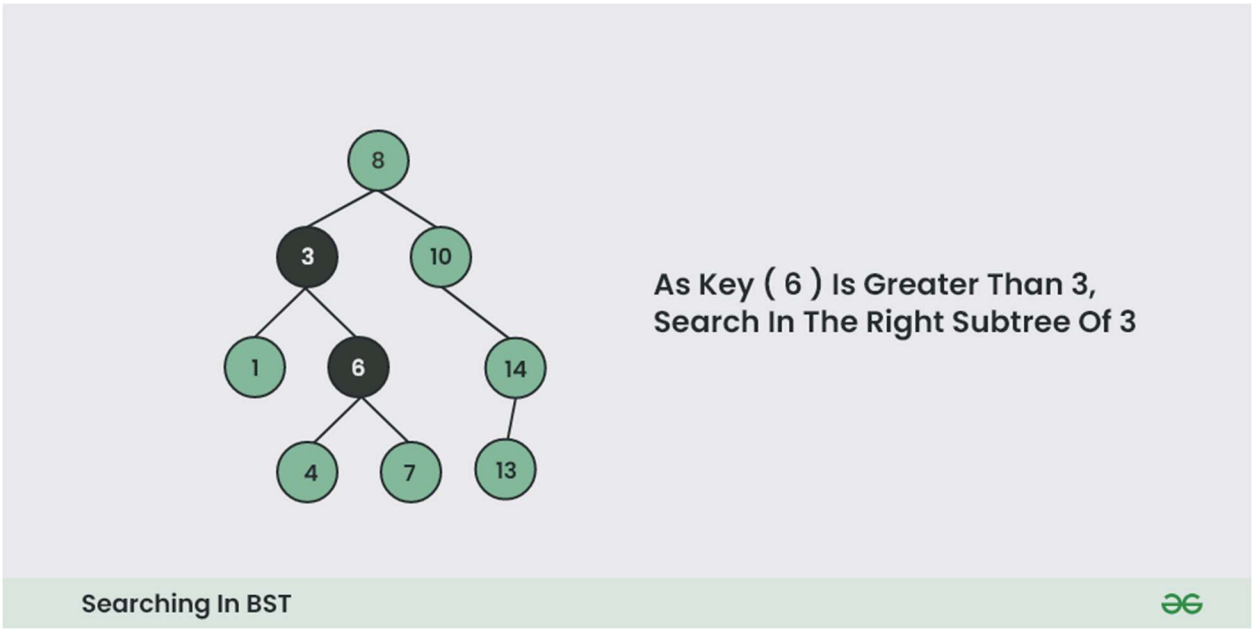
- We compare the value to be searched with the value of the root.
    - If it's equal we are done with the search if it's smaller we know that we need to go to the left subtree because in a binary search tree all the elements in the left subtree are smaller and all the elements in the right subtree are larger.
- Repeat the above step till no more traversal is possible
- If at any iteration, key is found, return True. Else False.

Illustration of searching in a BST:

See the illustration below for a better understanding:







// C++ function to search a given key in a given BST

```
#include <iostream>

using namespace std;
```

```
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
		= new struct node;
	temp->key = item;
	temp->left = temp->right = NULL;
	return temp;
}

// 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;
}

// Utility function to search a key in a 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);
}

// Driver Code
int main()
{
        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);
```

```
    // Key to be found
    int key = 6;

    // Searching in a BST
    if (search(root, key) == NULL)
        cout << key << " not found" << endl;
    else
        cout << key << " found" << endl;

    key = 60;

    // Searching in a BST
    if (search(root, key) == NULL)
        cout << key << " not found" << endl;
    else
        cout << key << " found" << endl;
    return 0;
}
```

Output

```
6 not found
```

```
60 found
```

Time complexity: O(h), where h is the height of the BST.
Auxiliary Space: O(h), where h is the height of the BST. This is because the maximum amount of space needed to store the recursion stack would be **h**.