We provide the recursive and iterative solution for your reference.

Here is a recursive solution:

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* public class TreeNode {  \* int val;  \* TreeNode left;  \* TreeNode right;  \* TreeNode(int x) { val = x; }  \* }  \*/  class Solution {  public TreeNode searchBST(TreeNode root, int target) {  if (root == null || root.val == target) {  return root;  }  if (target < root.val) {  return searchBST(root.left, target);  }  return searchBST(root.right, target);  }  } |

We can also solve this problem iteratively:

|  |
| --- |
| /\*\*  \* Definition for a binary tree node.  \* public class TreeNode {  \* int val;  \* TreeNode left;  \* TreeNode right;  \* TreeNode(int x) { val = x; }  \* }  \*/  class Solution {  public TreeNode searchBST(TreeNode root, int target) {  TreeNode cur = root;  while (cur != null && cur.val != target) {  if (target < cur.val) {  cur = cur.left;  } else {  cur = cur.right;  }  }  return cur;  }  } |

Let's discuss the time complexity and space complexity of the search operation in a BST whose height is h. Focus on the recursion solution first. In the worse case, the depth of our recursion is equal to the height of the tree. Therefore, the time complexity of the recursion solution is O(h). And taking system stack into consideration, the space complexity should be O(h) in the worst case as well.

What about the iterative solution? The time complexity will be equal to the loop time which is also O(h) while the space complexity is O(1) since we do not need system stack anymore in an iterative solution.

 Question:

If you do not know the height of the BST h but you are given the total number of nodes N of the BST, can you express the time complexity and space complexity using N instead of h?

Hint:

What's the difference of the relationship between N and h in the best case and the relationship in the worst case?