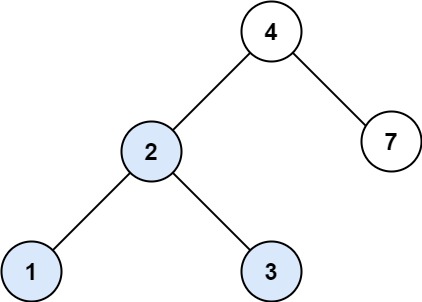
# Question

You are given the root of a binary search tree (BST) and an integer val.

Find the node in the BST that the node's value equals val and return the subtree rooted with that node. If such a node does not exist, return null.

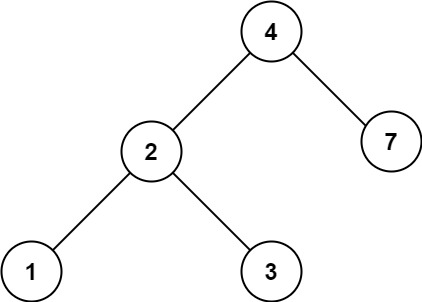
**Example 1:**



**Input:** root = [4,2,7,1,3], val = 2

**Output:** [2,1,3]

**Example 2:**



**Input:** root = [4,2,7,1,3], val = 5

**Output:** []

**Constraints:**

* The number of nodes in the tree is in the range [1, 5000].
* 1 <= Node.val <= 107
* root is a binary search tree.
* 1 <= val <= 107

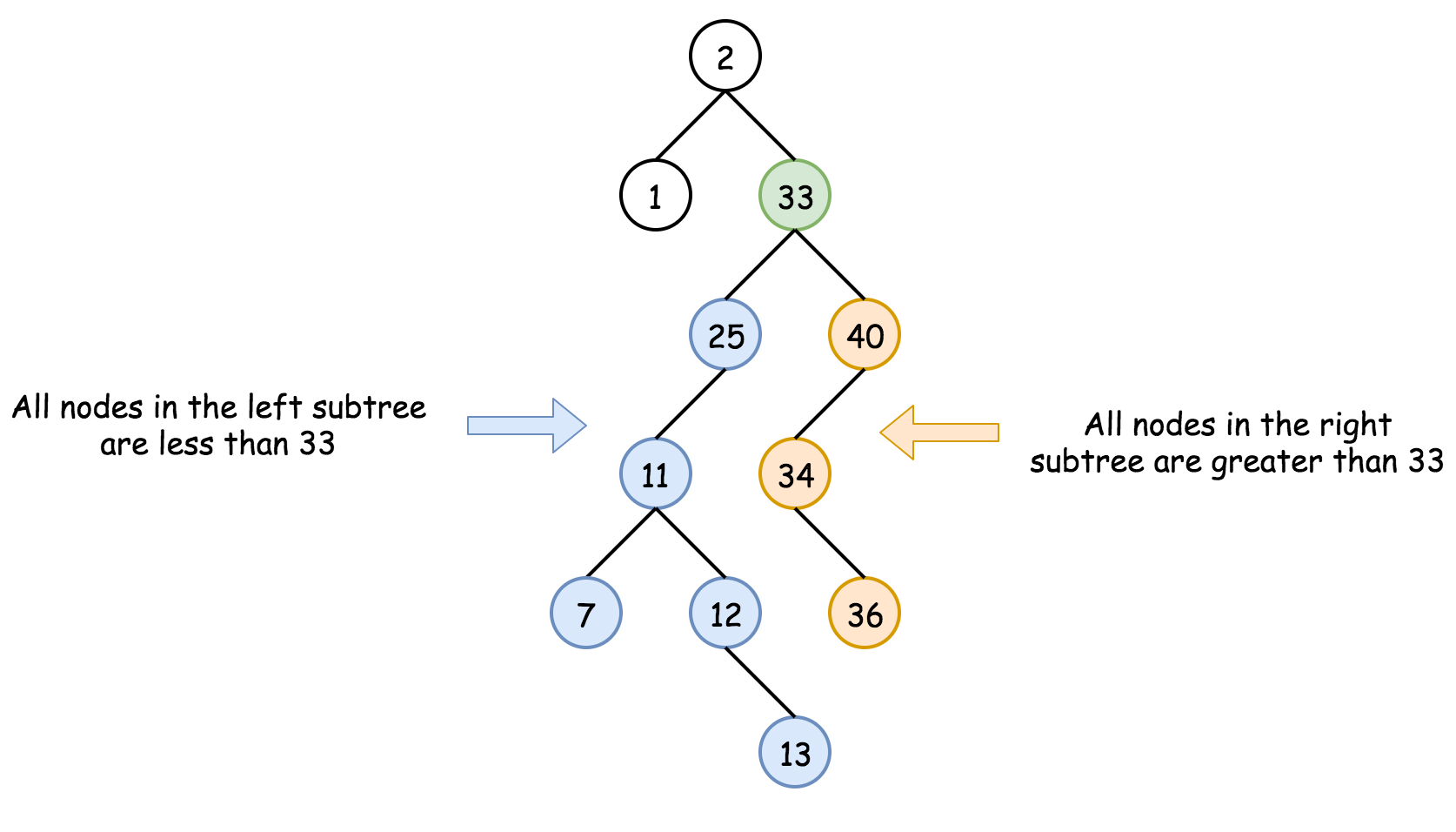
# Solution

#### **Binary Search Tree.**

Binary Search Tree is a binary tree where the key in each node

* is greater than any key stored in the left sub-tree,
* and less than any key stored in the right sub-tree.

Here is an example:



Such data structure provides the following operations in a logarithmic time:

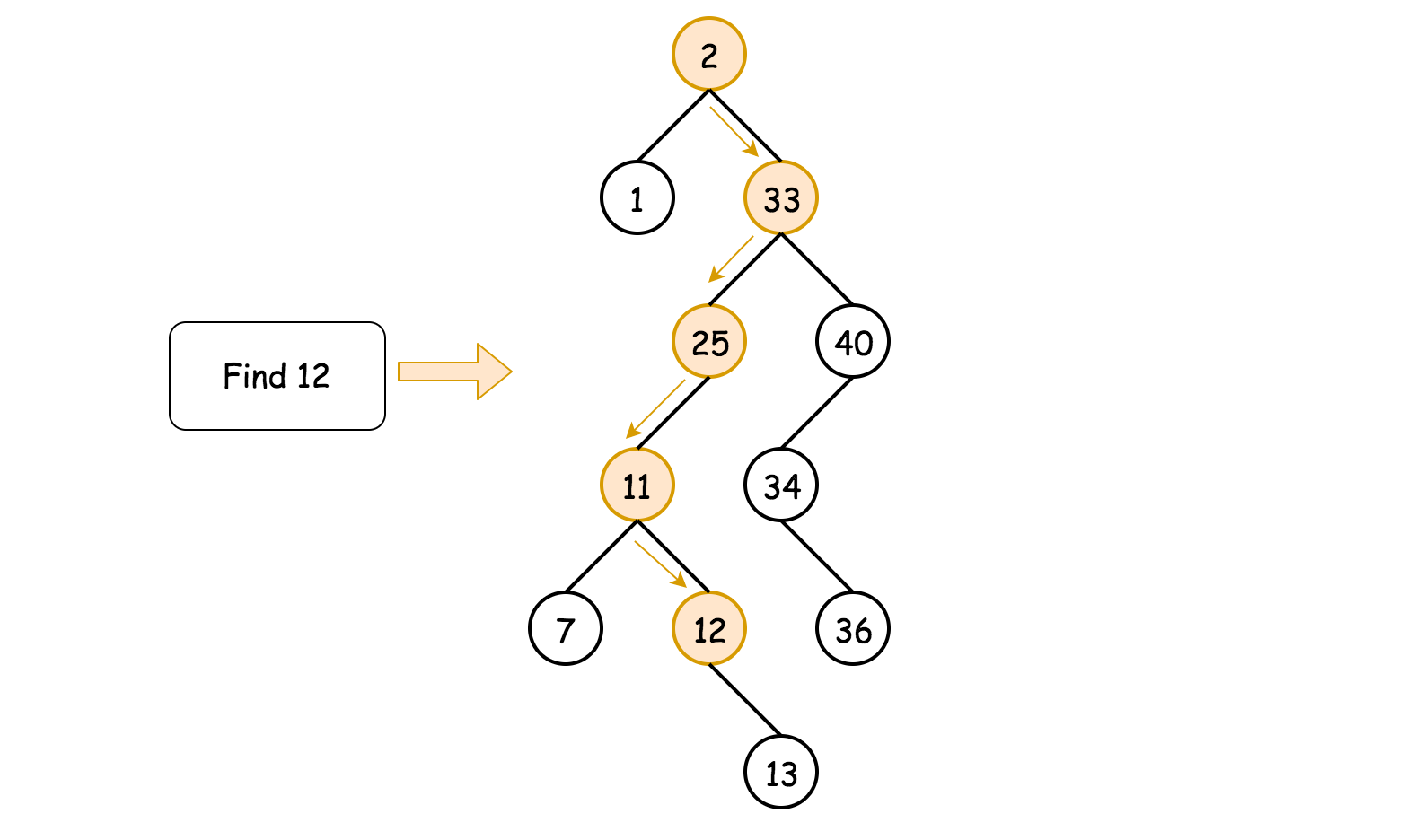
* Search.
* [Insert](https://leetcode.com/articles/insert-into-a-bst/).
* [Delete](https://leetcode.com/articles/delete-node-in-a-bst/).

#### **Approach 1: Recursion**

**Algorithm**

The recursion implementation is very straightforward:

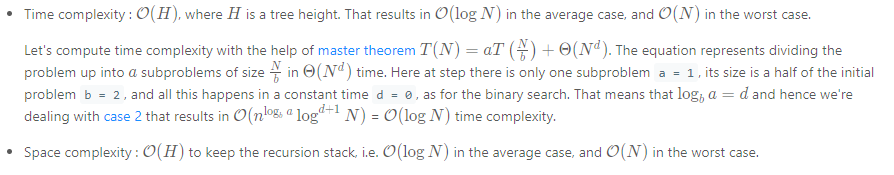
* If the tree is empty root == null or the value to find is here val == root.val - return root.
* If val < root.val - go to search into the left subtree searchBST(root.left, val).
* If val > root.val - go to search into the right subtree searchBST(root.right, val).
* Return root.



**Implementation**

|  |
| --- |
| class Solution {  public TreeNode searchBST(TreeNode root, int val) {  if (root == null || val == root.val) return root;  return val < root.val ? searchBST(root.left, val) : searchBST(root.right, val);  }  } |

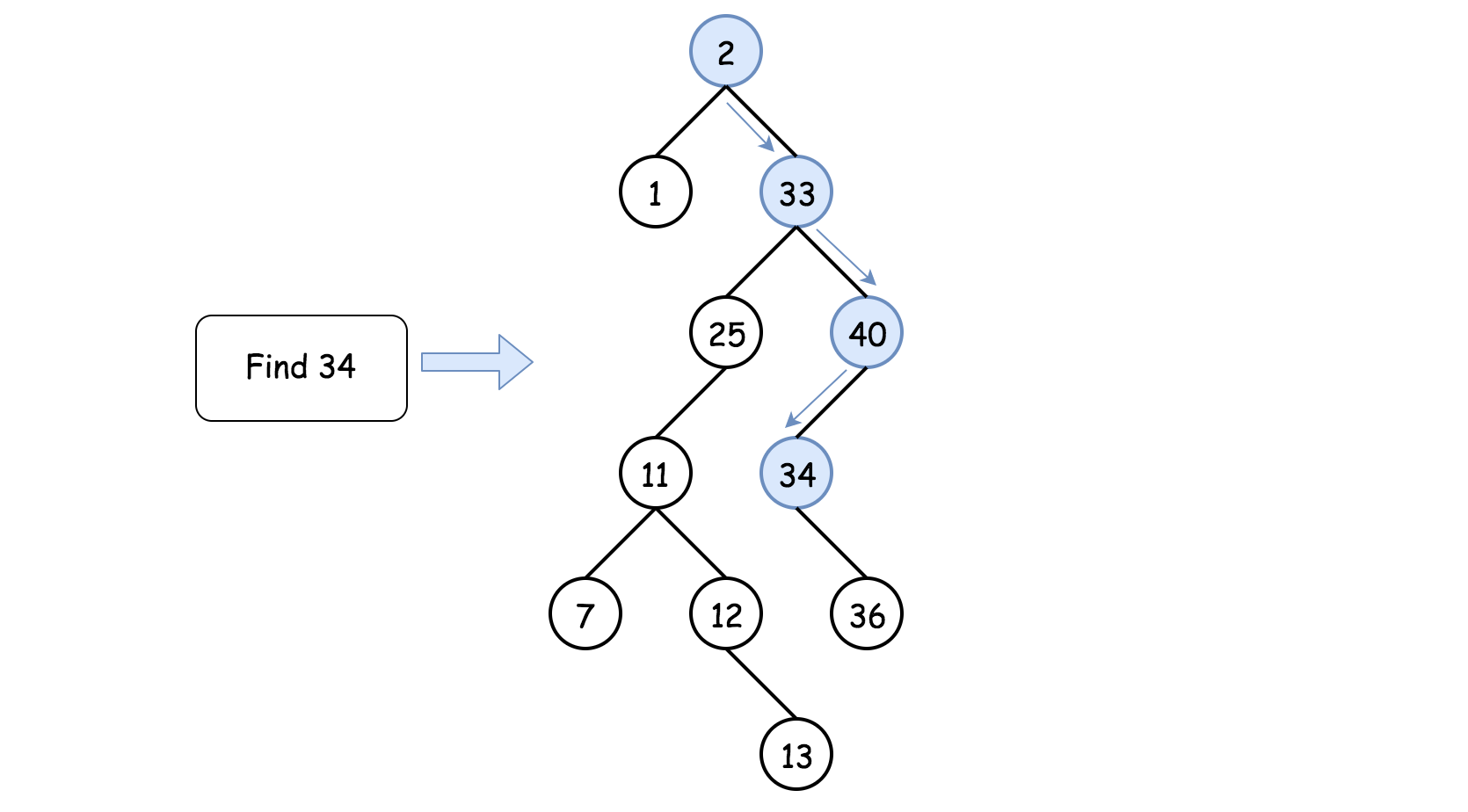
**Complexity Analysis**



#### **Approach 2: Iteration**

To reduce the space complexity, one could convert recursive approach into the iterative one:

* While the tree is not empty root != null and the value to find is not here val != root.val:
  + If val < root.val - go to search into the left subtree root = root.left.
  + If val > root.val - go to search into the right subtree root = root.right.
* Return root.



**Implementation**

|  |
| --- |
| class Solution {  public TreeNode searchBST(TreeNode root, int val) {  while (root != null && val != root.val)  root = val < root.val ? root.left : root.right;  return root;  }  } |

**Complexity Analysis**

