Summary

Time Complexity: O(H+k), where H is tree height. This complexity is defined by the stack, which contains at least H + k entries. Before starting to pop out one has to go down to a leaf. This results in O(logN+k) for the balanced tree and O(N+k) for a completely unbalanced tree with all the nodes in the left subtree.
Space Complexity: O(H+k), the same as for time complexity. O(N+k) in the worst case and O(logN+k) in the average case.
Problem Pattern

1.Think about different possible traversals of a tree. Note that the given tree is a Binary Search Tree. Can you utilize the properties of a BST to arrive at a solution?
2.What is special about inorder traversal of a BST? It is an array sorted in ascending order. Can you use this property?
3.Think about a recursive approach to this. What if you store the inorder traversal of the tree in an array and return the required element?
4.Think about the time and space complexity of your solution. Can you improve it if you solve the question iteratively instead, using a data structure that mimics recursion?
Problem Approach

1. The idea is to build an inorder traversal of BST which is an array sorted in the ascending order.

But with the help of stack, we could have a faster solution since we don't need to do the entire inorder traversal. We can stop at the kth element.

Problem Pseudocode

```
int kthSmallest(TreeNode root, int k) {
    Stack<TreeNode> stack
    while (true) {
        while (root is not null) {
            stack.push(root)
            root = root.left
        }
        root = stack.pop()
        if (--k == 0)
            return root.val
        root = root.right
}
```