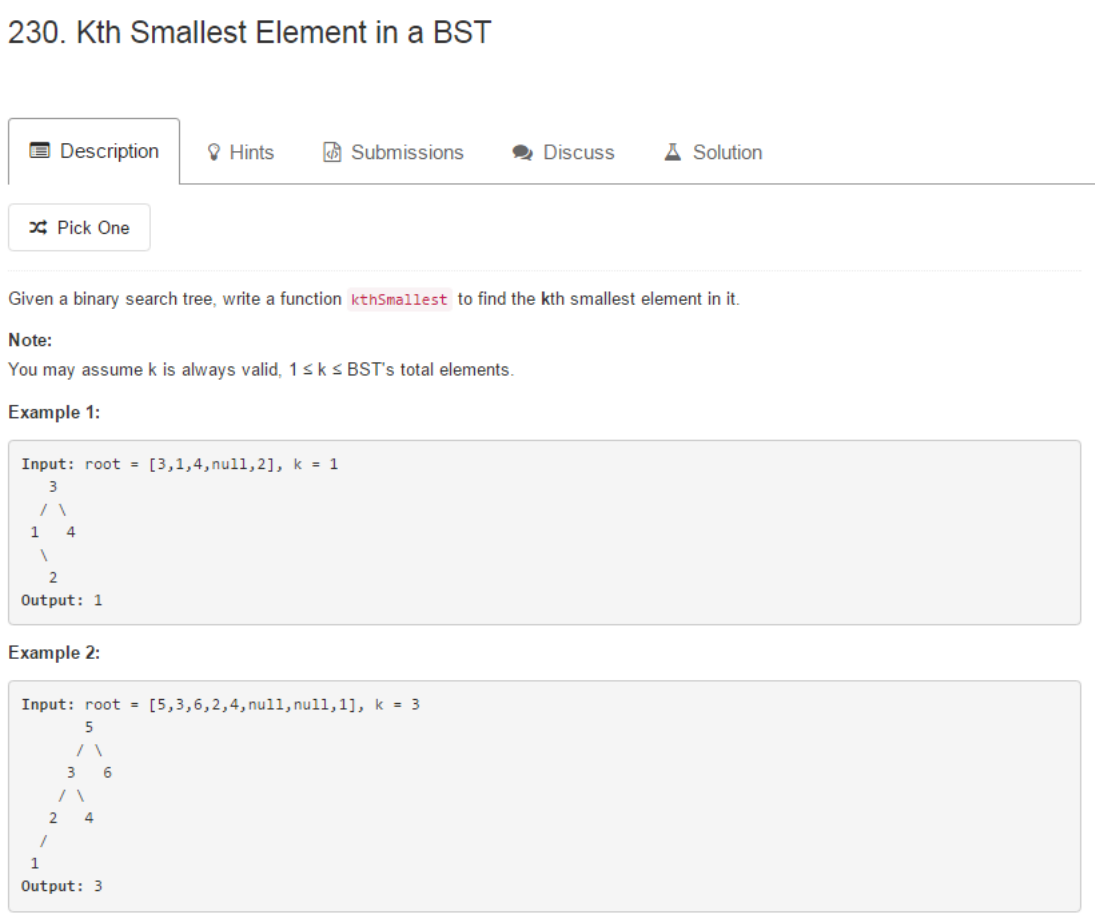


230. Kth Smallest Element in a BST

Given a binary search tree, write a function `kthSmallest` to find the **k**th smallest element in it.

Note:
You may assume k is always valid, 1 ≤ k ≤ BST's total elements.

Example 1:

```
Input: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
Output: 1
```

Example 2:

```
Input: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
Output: 3
```

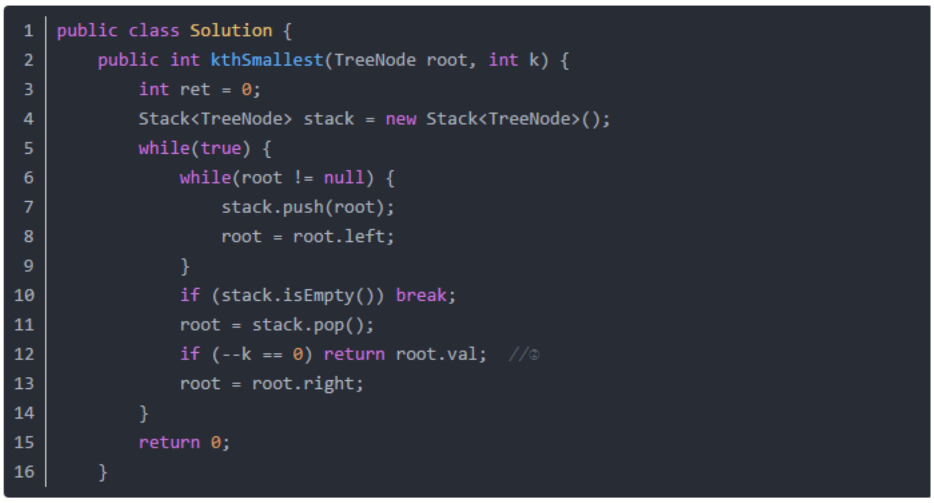
递归算法

```
public class L230 {

        public class TreeNode {
                int val;
                TreeNode left;
                TreeNode right;
                TreeNode(int x) { val = x; }
            }
      /*
       * 我们知道二分查找数(BST)的性质——任何一个节点的值均大于左子树的任意节点值，而小于右子树的任一节点值。
       * 那么这样就可以知道最小值的一个节点在树的最左端，最大值的一个节点在树的最右端。树从小到大顺序刚好满足树的中序遍历。因而，我们可以用中序遍历来处理。
       */
      private int count, res;
    public int kthSmallest(TreeNode root, int k) {
        if (root.left != null) kthSmallest(root.left, k);
        if (++count == k) res = root.val;      //①
        if (root.right != null) kthSmallest(root.right, k);
        return res;
    }

}
```

非递归



```java
public class Solution {
    public int kthSmallest(TreeNode root, int k) {
        int ret = 0;
        Stack<TreeNode> stack = new Stack<TreeNode>();
        while(true) {
            while(root != null) {
                stack.push(root);
                root = root.left;
            }
            if (stack.isEmpty()) break;
            root = stack.pop();
            if (--k == 0) return root.val;  //②
            root = root.right;
        }
        return 0;
    }
```