98. Validate Binary Search Tree
We will use in-order traverse to combine all node's value in bst. TC is O(n)

```
class Solution:
    def isValidBST(self, root: TreeNode) -> bool:
      self.pre = -float('inf')
      def dfs(node):
        if not node:
          return True
        if not dfs(node.left):
          return False

        if node.val <= self.pre:
          return False
        self.pre = node.val
        if not dfs(node.right):
          return False
        return True
      return dfs(root)
```

530. Minimum Absolute Difference in BST

```
class Solution:
    def getMinimumDifference(self, root: TreeNode) -> int:
        self.min = float('inf')
        self.pre = None
        def inorder(node):
          if not node:
            return None
          inorder(node.left)
          if self.pre is not None:
            self.min = min(self.min, node.val - self.pre)
          self.pre = node.val
          inorder(node.right)
        inorder(root)
        return self.min
```

700. Search in a Binary Search Tree

```
class Solution:
    def searchBST(self, root: TreeNode, val: int) -> TreeNode:
        def inorder(node):
          if not node:
            return None
          if node.val == val:
```

```
            return node
          elif node.val > val:
            return inorder(node.left)
          else:
            return inorder(node.right)
        return inorder(root)
```

701. Insert into a Binary Search Tree
We will try to find the slot we want to insert using recursion. TC is O(logn)

```
class Solution:
    def insertIntoBST(self, root: TreeNode, val: int) -> TreeNode:
        if not root:
          return TreeNode(val)
        def find(node, val):
          if node.val < val:
            if node.right:
              find(node.right, val)
            else:
              node.right = TreeNode(val)
          else:
            if node.left:
              find(node.left, val)
            else:
              node.left = TreeNode(val)
        find(root, val)
        return root
```

230. Kth Smallest Element in a BST
We will use stack to traverse our BST, TC is O(k)

```
class Solution:
    def kthSmallest(self, root: TreeNode, k: int) -> int:
        stack = [root]
        while root:
          stack.append(root)
          root = root.left

        while stack:
          node = stack.pop()
          k -= 1
          if k == 0:
            return node.val
          node = node.right
          while node:
```

```
stack.append(node)
node = node.left
```