暴力解法

Java代码

```java
class Solution {
    int res = Integer.MIN_VALUE;

    public int maxSumBST(TreeNode root) {
        if (root == null) {
            res = res > 0 ? res : 0;
            return res;
        }
        int[] state = isValidBST(root);
        if (state[0] == 1) {
            res = res > state[2] ? res : state[2];
        }
        maxSumBST(root.left);
        maxSumBST(root.right);
        return res;
    }
    public int[] isValidBST(TreeNode root) {
        int[] state = new int[3];
        state[0] = 0;
        state[1] = Integer.MIN_VALUE;
        state[2] = 0;
        helper(root, state);
        return state;
    }

    private void helper(TreeNode root, int[] state) {
        if (root == null) {
            state[0] = 1;
            return;
        }
        helper(root.left, state);
        if (root.val <= state[1] || state[0] == 0) {
            state[0] = 0;
            state[0] = 0;
            return;
        }
        state[1] = root.val;
        state[2] = state[2] + root.val;
        helper(root.right, state);
    }
}
```

优化解法

Java代码

```java
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<Integer>();
        inorder(root, res);
        return res;
    }

    public void inorder(TreeNode root, List<Integer> res) {
        if (root == null) {
            return;
        }
        inorder(root.left, res);
        res.add(root.val);
        inorder(root.right, res);
    }
}
```

最优解法

Java代码

```java
class Solution {
       class Result {
        int min;
        int max;
        int sum;
        boolean isBST;

        public Result(int min, int max, int sum, boolean isBST) {
            this.min = min;
            this.max = max;
            this.sum = sum;
            this.isBST = isBST;
        }
    }

    public int maxSumBSTPostOrder(TreeNode root) {
        int[] max = new int[1];
        traverse(root, max);
        return max[0];
    }
    private Result traverse(TreeNode root, int[] max) {
        if (root == null) {
            return null;
        }
        Result leftResult = traverse(root.left, max);
        Result rightResult = traverse(root.right, max);

        if (leftResult != null && ((!leftResult.isBST) || leftResult.max >=
root.val)) {
            return new Result(Integer.MIN_VALUE, Integer.MAX_VALUE, 0, false);
        }
        if (rightResult != null && ((!rightResult.isBST) || rightResult.min <=
root.val)) {
            return new Result(Integer.MIN_VALUE, Integer.MAX_VALUE, 0, false);
        }

        int nsum = leftResult == null ? 0 : leftResult.sum;
        nsum += rightResult == null ? 0 : rightResult.sum;
        nsum += root.val;

        int minval = leftResult == null ? root.val : leftResult.min;
        int maxval = rightResult == null ? root.val : rightResult.max;
        max[0] = Math.max(max[0], nsum);
        return new Result(minval, maxval, nsum, true);
    }
}
```

C++代码

```
class Solution {
public:
    int maxsum = 0;
    int maxSumBST(TreeNode* root) {
        dfs(root);
        return maxsum;
    }
    vector<int> dfs(TreeNode* root) {
        if (!root) return {true, INT_MAX, INT_MIN, 0};
        auto lArr = dfs(root->left);
        auto rArr = dfs(root->right);
        int sum = 0, curmax, curmin;
        if (!lArr[0] || !rArr[0] || root->val >= rArr[1] || root->val <=
lArr[2]) {
            return {false, 0, 0, 0};
        }
        curmin = root->left ? lArr[1] : root->val;
        curmax = root->right ? rArr[2] : root->val;
        sum += (root->val + lArr[3] + rArr[3]);
        maxsum = max(maxsum, sum);
        return {true, curmin, curmax, sum};
    }
};
```

Python代码

```
class Solution:
    def maxSumBST(self, root: TreeNode) -> int:
        self.max_value = 0
        self.helper(root)
        return self.max_value

    def helper(self, root):
        # 返回三个变量
        # 分别为【以当前节点为根节点的二叉搜索树的键值和】，【上界】，【下界】
        if not root:
            return 0, 5e4, -5e4
        value1, min_value1, max_value1 = self.helper(root.left)
        value2, min_value2, max_value2 = self.helper(root.right)
        if max_value1 < root.val and min_value2 > root.val:
            # 满足二叉搜索树条件
            self.max_value = max(self.max_value, value1 + value2 + root.val)
            return value1 + value2 + root.val, min(min_value1, root.val),
max(max_value2, root.val)
        # 说明该节点无法构成二叉搜索树，返回恒不成立的条件，一直返回到顶
        return root.val, -5e4, 5e4
```