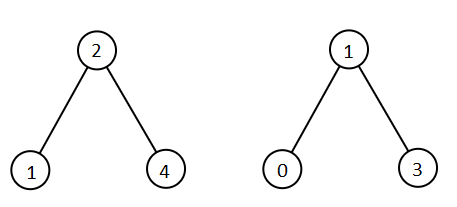
# 描述

给你 root1和root2这两棵二叉搜索树。

请你返回一个列表，其中包含两棵树中的所有整数并按升序排序。.

**示例 1：**



输入：root1 = [2,1,4], root2 = [1,0,3]

输出：[0,1,1,2,3,4]

**示例 2：**

输入：root1 = [0,-10,10], root2 = [5,1,7,0,2]

输出：[-10,0,0,1,2,5,7,10]

**示例 3：**

输入：root1 = [], root2 = [5,1,7,0,2]

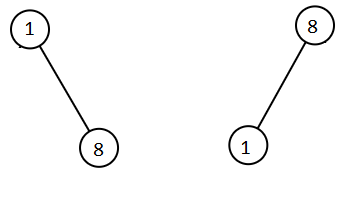
输出：[0,1,2,5,7]

**示例 4：**

输入：root1 = [0,-10,10], root2 = []

输出：[-10,0,10]

**示例 5：**



输入：root1 = [1,null,8], root2 = [8,1]

输出：[1,1,8,8]

**提示：**

每棵树最多有 5000 个节点。

每个节点的值在 [-10^5, 10^5] 之间。

# 分析

## 方法一：遍历 + 排序

我们可以想到的最简单的方法是，对两棵树进行任意形式的遍历（深度优先搜索、广度优先搜索、前序遍历、中序遍历、后序遍历），并将遍历到的所有元素放入一个数组中，最后对这个数组进行排序即可。

## 方法二：中序遍历 + 归并排序

方法一中并没有用到二叉搜索树本身的性质。如果我们对二叉搜索树进行中序遍历，就可以直接得到树中所有元素升序排序后的结果。因此我们可以对两棵树分别进行中序遍历，得到数组 v1 和 v2，它们分别存放了两棵树中的所有元素，且均已有序。在这之后，我们通过归并排序的方法对 v1 和 v2 进行排序，就可以得到最终的结果。

要解决这个问题，我们需要从两棵棵二叉搜索树中提取所有整数，并将它们按升序排序后返回。由于二叉搜索树的中序遍历结果是升序序列，我们可以利用这一特性高效地解决问题。

解题思路：

1、中序遍历提取元素：分别对两棵二叉搜索树进行中序遍历，得到两个升升序数组。这是因为二叉搜索树的特性保证了中序遍历结果为有序序列。

2、归并排序合并数组：由于两个数组都是升序的，我们可以使用归并排序的合并步骤，将两个有序数组合并为一个有序数组。这种方法比先合并再整体排序更高效。

代码：

```cpp

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode() : val(0), left(nullptr), right(nullptr) {}

\* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

\* TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

\* };

\*/

class Solution {

public:

vector<int> getAllElements(TreeNode\* root1, TreeNode\* root2) {

// 存储两棵树的中序遍历结果（均为升序）

vector<int> vals1, vals2;

inorder(root1, vals1);

inorder(root2, vals2);

// 归并两个升序数组

return merge(vals1, vals2);

}

private:

// 中序遍历二叉搜索树，将节点值按升序存入数组

void inorder(TreeNode\* root, vector<int>& vals) {

if (root == nullptr) return;

inorder(root->left, vals); // 遍历左子树

vals.push\_back(root->val); // 记录当前节点值

inorder(root->right, vals); // 遍历右子树

}

// 归并两个升序数组，返回合并后的升序数组

vector<int> merge(vector<int>& a, vector<int>& b) {

vector<int> res;

int i = 0, j = 0;

// 双指针遍历两个数组，按顺序合并

while (i < a.size() && j < b.size()) {

if (a[i] < b[j]) {

res.push\_back(a[i++]);

} else {

res.push\_back(b[j++]);

}

}

// 处理剩余元素

while (i < a.size()) {

res.push\_back(a[i++]);

}

while (j < b.size()) {

res.push\_back(b[j++]);

}

return res;

}

};

解释：

1、中序遍历：inorder函数通过递归遍历二叉搜索树的左子树、记录当前节点值、遍历右子树，将节点值按升序存入数组。对于两棵树分别执行此操作，得到两个升序数组vals1和vals2。这一步的时间复杂度为O(n + m)（n和m分别为两棵树的节点数），空间复杂度为`O(n + m)`（存储所有节点值）。

2、归并排序合并：merge函数使用双指针技术合并两个升序数组。初始时，两个指针分别指向两个数组的起始位置，比较指针所指元素的大小，将较小的元素加入结果数组，并移动对应指针。当一个数组遍历完毕后，将另一个数组的剩余元素依次加入结果数组。这一步的时间复杂度为O(n + m)，空间复杂度为O(n + m)（存储合并后的结果）。

通过这种方法，我们充分利用了二叉搜索树的特性和归并排序的高效性，整体时间复杂度为O(n + m)，是解决该问题的最优方案之一。