Editorial

X

Approach

Here is the algorithm:

1. Construct inorder traversal of the tree. It should be an almost sorted list where only two elements are swapped.
2. Identify two swapped elements x and y in an almost sorted array in linear time.
3. Traverse the tree again. Change value x to y and value y to x.

Complexity Analysis :

Time complexity : O(N). To compute inorder traversal takes O(N) time, to identify and to swap back swapped nodes O(1) in the best case and O(N) in the worst.
Space complexity : O(N) since we keep inorder traversal nums with N elements.

C++ Code

```
class Solution
{
private:
    void inorder(TreeNode *root, vector<int> &nums)
    {
        if (root == NULL)
            return;
        inorder(root->left, nums);
        nums.push_back(root->val);
        inorder(root->right, nums);
    }
    pair<int, int> findTwoSwapped(vector<int> nums)
    {
        int n = nums.size();
        int x = -1, y = -1;
        for (int i = 0; i < n - 1; ++i)
        {
            if (nums[i + 1] < nums[i])
            {
                y = nums[i + 1];
                // first swap occurence
                if (x == -1)
                    x = nums[i];
                // second swap occurence
                else
                    break;
            }
        }
        return {x, y};
    }

public:
    void recoverTree(TreeNode *root)
    {
        vector<int> nums;
        inorder(root, nums);
        pair<int, int> swapped = findTwoSwapped(nums);
        cout << min(swapped.first, swapped.second) << " " << max(swapped
    }
};
```

Java Code

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

  public int[] findTwoSwapped(List<Integer> nums) {
    int n = nums.size();
    int x = -1, y = -1;
    for(int i = 0; i < n - 1; ++i) {
      if (nums.get(i + 1) < nums.get(i)) {
        y = nums.get(i + 1);
        // first swap occurence
        if (x == -1) x = nums.get(i);
        // second swap occurence
        else break;
      }
    }
    return new int[]{x, y};
  }



  public void recoverTree(TreeNode root) {
    List<Integer> nums = new ArrayList();
    inorder(root, nums);
    int[] swapped = findTwoSwapped(nums);
    System.out.println(Math.min(swapped[0], swapped[1]) + " " + Math.max
  }
}
```

Python Code

```
class Solution:
    def recoverTree(self, root: TreeNode):
        def inorder(r: TreeNode) -> List[int]:
            return inorder(r.left) + [r.val] + inorder(r.right) if r els

        def find_two_swapped(nums: List[int]) -> (int, int):
            n = len(nums)
            x = y = -1
            for i in range(n - 1):
                if nums[i + 1] < nums[i]:
                    y = nums[i + 1]
                    # first swap occurence
                    if x == -1:
                        x = nums[i]
                    # second swap occurence
                    else:
                        break
            return x, y

        nums = inorder(root)
        x, y = find_two_swapped(nums)
        return x, y
```