2 Example 2: Input: [3,1,4,null,null,2] 3 Output: [2,1,4,null,null,3] 2 3 Follow up: • A solution using O(n) space is pretty straight forward.

• Could you devise a constant space solution? Solution

solution serves to identify and discuss all subproblems. It's known that inorder traversal of BST is an array sorted in the ascending order. Here is how one could

Python Java def inorder(r: TreeNode) -> List[int]:

Python Java def find_two_swapped(nums: List[int]) -> (int, int): n = len(nums)

3

x = nums[i]9 # second swap occurence 10 else: 11 12 break 13 return x, y

Inorder traversal:

an array sorted in the ascending order where two elements are swapped

Traverse the tree [1, 3, 2, 4]and change node values [1, 2, 3, 4]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. Implementation

27

```
• Merge steps 1 and 2, i.e. identify swapped nodes during the inorder traversal.
   • Swap node values.
The difference in-between the following approaches is in a chosen method to implement inorder traversal:
   • Approach 2 : Iterative.
   • Approach 3: Recursive.

    Approach 4: Morris.

                                 Inorder Traversal
                     Recursive
                                                             Morris
           = the simplest one to write
                                                       = constant space
```

stack = []

15 16 17 18 19

Iterative approach 2 could be converted into recursive one. Recursive inorder traversal is extremely simple: follow Left->Node->Right direction, i.e. do the recursive call for the left child, then do all the business with the node (= if the node is the swapped one or not), and

DFS Postorder

2

postorder(root.left) +

[root.val]

if root else []

postorder(root.right) +

BFS

5

Сору

1/11

iterations

with the queue

4

Approach 4: Morris Inorder Traversal

Copy Python Java class Solution: 1 def recoverTree(self, root): 3 4 :type root: TreeNode

In the 'loop' cases it could be equal to the node itself predecessor == root.

set link predecessor.right = root 26 # and go to explore left subtree 27 if predecessor.right is None: • Time complexity : $\mathcal{O}(N)$ since we visit each node up to two times. • Space complexity : $\mathcal{O}(1)$. O Previous Next 👀 Comments: 17 Sort By ▼ Type comment here... (Markdown is supported) Preview Post jolyon129 ★ 126 ② July 22, 2019 5:51 PM Literally the best article I've seen on leetcode. Thx! 61 A V C Share Reply khl7 ★ 51 ② July 7, 2019 4:01 AM A Report Very nicely written article. Thank you! 10 A V C Share Reply

- dilit * 37 April 4, 2020 2:47 AM @liaison and @andvary: Solution 1 does not need a 3rd method. You just store the node references directly in the array. In the find.. method, add .val to compare the nodes. Swap vals when the the nodes are found.

Read More

A Report

A Report

A Report

2

x = y = -1

- Here two nodes are swapped, and hence inorder traversal is an almost sorted array where only two elements are swapped. To identify two swapped elements in a sorted array is a classical problem that could be solved in linear time. Here is a solution code
- if nums[i + 1] < nums[i]:</pre> 5 6 y = nums[i + 1]7 # first swap occurence 8 if x == -1:
- Java 2 3 4
 - swapped nodes : $\mathcal{O}(1)$ in the best case and $\mathcal{O}(N)$ in the worst.

Iterative

= the best time performance

- Here we construct inorder traversal by iterations and identify swapped nodes at the same time, in one pass.
- time, drastically reducing the time needed for step 3. Inorder traversal
- x = y = pred = None8 9 while stack or root: 10 while root: 11 stack.append(root) 12 root = root.left

root = root.right

- nonlocal x, y, pred 8 if root is None: 9 10 return 11 find_two_swapped(root.left) 12 13

def recoverTree(self, root):

additional memory is allowed?

5 :rtype: void Do not return anything, modify root in-place instead. 6 # predecessor is a Morris predecessor.

If there is no link predecessor.right = root --> set it.

If there is a link predecessor.right = root --> break it.

while predecessor.right and predecessor.right != root:

- - AM_123 ★ 22 ② October 21, 2019 5:36 AM Best article read!!
 - qaref18 ★ 205 ② September 1, 2019 7:04 AM 5 stars rated article
 - 1 A V C Share Reply **SHOW 2 REPLIES** Seayahh 🛊 100 🗿 August 23, 2019 8:59 AM

- - Approach 1: Sort an Almost Sorted Array Where Two Elements Are Swapped Intuition
 - compute an inorder traversal return inorder(r.left) + [r.val] + inorder(r.right) if r else []

Let's start from straightforward but not optimal solution with a linear time and space complexity. This

- for i in range(n 1): 4
- [1, 3, 2, 4]Identify two swapped elements
- :rtype: void Do not return anything, modify root in-place instead. 5 6 def inorder(r: TreeNode) -> List[int]: 7 return inorder(r.left) + [r.val] + inorder(r.right) if r else [] 8 9 def find_two_swapped(nums: List[int]) -> (int, int): 10 n = len(nums)

for i in range(n - 1):

else:

return x, y

if nums[i + 1] < nums[i]:

first swap occurence

second swap occurence

x = nums[i]

y = nums[i + 1]

if x == -1:

break

def recover(r: TreeNode, count: int):

count -= 1

if r.val == x or r.val == y:

r.val = y if r.val == x else x

- 1. Construct inorder traversal. 2. Find swapped elements in an almost sorted array where only two elements are swapped. 3. Swap values of two nodes. Now we will discuss three more approaches, and basically they are all the same :
- Approach 2: Iterative Inorder Traversal
 - Stack
 - y = root if x is None: x = predelse: break pred = root

:rtype: void Do not return anything, modify root in-place instead.

- is a rightmost leaf. ullet Space complexity : up to $\mathcal{O}(H)$ to keep the stack where H is a tree height.
- 5 5 3 2 3 2

Traversal = [1, 2, 3, 4, 5]

- :type root: TreeNode :rtype: void Do not return anything, modify root in-place instead. 5 6 def find_two_swapped(root: TreeNode): 7 if pred and root.val < pred.val:</pre> 14 y = root # first swap occurence if x is None: x = pred# second swap occurence return pred = root find_two_swapped(root.right) x = y = pred = None25 find_two_swapped(root) 26 x.val, y.val = y.val, x.val **Complexity Analysis**
- There is no link? Set it and go to the left subtree. • There is a link? Break it and go to the right subtree. There is one small issue to deal with: what if there is no left child, i.e. there is no left subtree? Then go straightforward to the right subtree. root = 3predecessor = None "true" predecessor pred = None

There is lef

1. Find predecessor

2. Manage link predecessor.right = root

M

predecessor.right = root . So one starts from the node, computes its predecessor and verifies if the link

- **Complexity Analysis**
- In the iterative solution, I don't get the part of y = root;if (x == null) x = pred;else break:
 - 2 A V C Share Reply yakkaladevi 🖈 5 🗿 April 16, 2020 1:19 AM Real smooth article.
 - Very clear and logic. Thank you! :) 1 A V C Share Reply
 - HumanAfterA11 ★ 19 ② June 16, 2020 5:23 AM why is stack not recommended here?
 - (1 2)

- Iterative inorder traversal is simple: go left as far as you can, then one step right. Repeat till the end of nodes in the tree. To identify swapped nodes, track the last node pred in the inorder traversal (i.e. the predecessor of the current node) and compare it with current node value. If the current node value is smaller than its predecessor pred value, the swapped node is here. There are only two swapped nodes here, and hence one could break after having the second node identified.
 - Don't use Stack in Java, use ArrayDeque instead. Python Java class Solution: def recoverTree(self, root: TreeNode): 3 5 6 7
 - Approach 3: Recursive Inorder Traversal
 - 3 [root.val] + preorder(root.left) + preorder(root.right) if root else [] **Implementation** Java 1
 - is a rightmost leaf. ullet Space complexity : up to $\mathcal{O}(H)$ to keep the recursion stack, where H is a tree height.

 - 16 17 18 19 20 21 22 23 24
 - - 1 A V C Share Reply RainNight ★ 2 ② December 23, 2019 7:20 AM

Сору

Copy Copy

Сору

- Time complexity : $\mathcal{O}(N)$. To compute inorder traversal takes $\mathcal{O}(N)$ time, to identify and to swap back

Сору

- Doing so, one could get directly nodes (and not only their values), and hence swap node values in $\mathcal{O}(1)$
 - root = stack.pop() if pred and root.val < pred.val:
- x.val, y.val = y.val, x.val **Complexity Analysis** ullet Time complexity : $\mathcal{O}(1)$ in the best case, and $\mathcal{O}(N)$ in the worst case when one of the swapped nodes

DFS Inorder

3

inorder(root.left) +

inorder(root.right)

[root.val] +

if root else []

- 15 16 17 18 19
- We discussed already iterative and recursive inorder traversals, which both have great time complexity though use up to $\mathcal{O}(H)$ to keep stack. We could trade in performance to save space. The idea of Morris inorder traversal is simple: to use no space but to traverse the tree. How that could be even possible? At each node one has to decide where to go: left or right, traverse left subtree or traverse right subtree. How one could know that the left subtree is already done if no
- Implementation

pred is a 'true' predecessor,

x = y = predecessor = pred = None

If there is a left child

then compute the predecessor.

predecessor = root.left

while root:

if root.left:

the previous node in the inorder traversal.

Predecessor node is one step left

predecessor = predecessor.right

and then right till you can.

- Rate this article: * * * * *
- liang54 ★ 163 ② September 15, 2019 1:28 AM
 - 2 A V C Share Reply

- **Algorithm** Here is the algorithm:
 - Python class Solution: def recoverTree(self, root: TreeNode): x = y = -1
- ullet Space complexity : $\mathcal{O}(N)$ since we keep inorder traversal $rac{ extsf{nums}}{ extsf{nums}}$ with N elements. What Is Coming Next In approach 1 we discussed three easy subproblems of this hard problem:
- Iterative and recursive approaches here do less than one pass, and they both need up to $\mathcal{O}(H)$ space to keep stack, where H is a tree height. Morris approach is two pass approach, but it's a constant-space one.

- Implementation
- 14 20 21 22 23
- then do the recursive call for the right child. On the following figure the nodes are numerated in the order you visit them, please follow 1-2-3-4-5 to compare different DFS strategies. DFS Preorder Node -> Left -> Right Left -> Node -> Right Left -> Right -> Node Node -> Left -> Right
- 3 4
- ullet Time complexity : $\mathcal{O}(1)$ in the best case, and $\mathcal{O}(N)$ in the worst case when one of the swapped nodes
- 7 8 9 10 11 12 13 14 15
- - SHOW 5 REPLIES
 - In approach 2, why "Don't use Stack in Java, use ArrayDeque instead."?

- When swapped nodes are known, one could traverse the tree again and swap their values.
- 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 **Complexity Analysis**
- Intuition
- - 13

- 2

Python

class Solution:

- 20 21 22 23 24
- The idea of Morris algorithm is to set the temporary link between the node and its predecessor:

is present.

- - 6 A V C Share Share