**226. Invert Binary Tree**

<https://leetcode.com/problems/invert-binary-tree/>

1. **Listen**

**Problem Statement:**

Given the **root** of a **binary tree**, **invert** the tree, and **return *its root***.

**Input:**

**root** of binary tree

**Goal:**

**invert** the binary tree

**Return:**

root of inverted binary tree

1. **Example**

A picture containing text, clipart

Description automatically generated

**Input:** root = [2,1,3]

**Output:** [2,3,1]

A picture containing clock, clipart, watch

Description automatically generated

**Input:** root = [4,2,7,1,3,6,9]

**Output:** [4,7,2,9,6,3,1]

**Constraints:**

* The number of nodes in the tree is in the range [0, 100].
* -100 <= Node.val <= 100

**Test Cases:**

* Empty/Null root
* Unbalanced tree
* Perfect/unperfect tree

1. **Brute Force**

**Solution 1:**

Recursively swap each node’s left and right children.

This can be accomplished with DFS tail recursion.

O(n) time and O(logn) space

Bound to the application stack, which means that it is not scalable past that.

If we overflow the program stack with a large enough problem size, then it will crash the application.

1. **Optimize**

**Solution 2:**

To avoid being bound to the application stack, we can use an actual Stack data structure.

While the runtime and space will be the same, O(n) time and O(logn) space, this solution is much more scalable

**Solution 3:**

An alternative solution would be to do a BFS (level-order) traversal of the tree.

1. **Walkthrough**

Initialize a stack and push the root node onto it

while the stack is not empty

pop the root

push the left and right children of the current root onto the stack

swap the roots left and right children

1. **Implement**

**Recursive DFS**

private TreeNode invertTreeHelper(TreeNode root)

{

if(root == null) return null;

TreeNode temp = root.left;

root.left = invertTreeHelper(root.right);

root.right = invertTreeHelper(temp);

return root;

}

**Iterative DFS**

private TreeNode invertTreeIterative(TreeNode root)

{

Stack<TreeNode> stack = new Stack<>();

stack.push(root);

while(!stack.isEmpty())

{

TreeNode current = stack.pop();

if(current.right != null) stack.push(current.right);

if(current.left != null) stack.push(current.left);

TreeNode temp = current.left;

current.left = current.right;

current.right = temp;

}

return root;

}

**Iterative BFS**

1. **Test**

Empty/Null root

* added null test case at beginning of function
* works now

Unbalanced tree

* works even with null values in an unbalanced tree