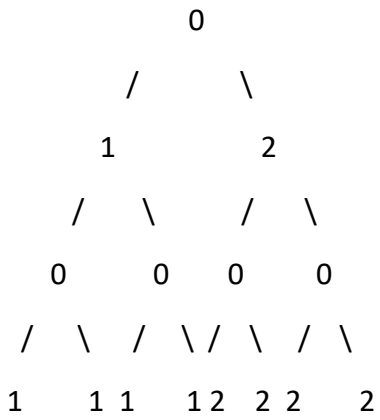
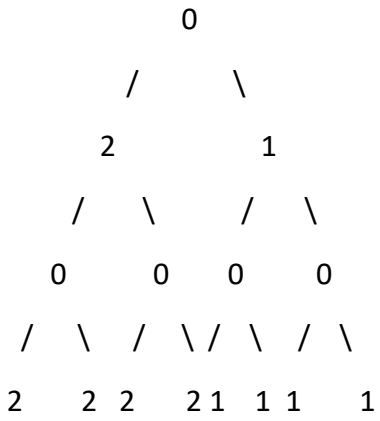
Reverse Odd Levels of Binary Tree LeetCode

Given the root of a **perfect** binary tree, reverse the node values at each **odd** level of the tree.

Example:



Output:



Approach 1: Function to reverse odd levels of a binary tree using recursion

- The **reverseOddLevelsRecursive** function reverses the values of nodes at odd levels in the binary tree recursively.
- It initializes the level as 1 and calls the helper function **reverseTree**.
- The **reverseTree** function takes two nodes, **leftNode** and **rightNode**, and the current level as parameters.
- In the **reverseTree** function, if the current level is odd (determined by checking the least significant bit), it swaps the values of **leftNode** and **rightNode**.
- Then, it recursively calls **reverseTree** on the left and right subtrees with an incremented level.
- Finally, it returns the root of the modified binary tree.

Time Complexity: O(N), where N is the number of nodes in the binary tree. You visit each node once.

Space Complexity: O(H), where H is the height of the binary tree due to the function call stack.

Approach 2: Function to reverse odd levels of a binary tree using an iterative approach

- The **reverseOddLevelsIterative** function reverses the values of nodes at odd levels in the binary tree using an iterative level-order traversal.
- It uses a queue (**q**) to perform level-order traversal and a vector (**currQueue**) to temporarily store values at odd levels.
- The function initializes the level as 0.
- In each level of the loop, it processes nodes and checks if the current level is odd.
- If the level is odd, it sets the value of the current node to the value from **currQueue**.
- It then pushes the left and right children onto the queue for further processing.
- After processing each level, it updates **currQueue** with values at the next odd level.
- The loop continues until the queue is empty.
- Finally, it returns the root of the modified binary tree.

Time Complexity: O(N), where N is the number of nodes in the binary tree. You visit each node once.

Space Complexity: O(W), where W is the maximum width of the binary tree at any level due to the queue and vector.