Flatten a Binary Tree To a Linked List [LeetCode](https://leetcode.com/problems/flatten-binary-tree-to-linked-list/description/)

Given the root of a binary tree, flatten the tree into a "linked list":

* The "linked list" should use the same Node class where the right child pointer points to the next node in the list and the left child pointer is always null.
* The "linked list" should be in the same order as a preorder traversal of the binary tree.

Example:

1

/ \

2 6

/ \ / \

3 4 7 8

/ / / \

5 9 10 11

Output: 1 → 2 → 3 → 5 → 4 → 6 → 7 → 9 → 8 → 10 → 11

**Approach 1: Flatten a binary tree using a recursive approach**

* A public function **flattenRecursive** is used to initiate the flattening process.
* It utilizes the helper function **flattenHelper** that recursively flattens the binary tree into a linked list.
* In the **flattenHelper** function, a post-order traversal is used:
  + Recursively flatten the right subtree first, then the left subtree.
  + Connect the current node to the previously flattened portion.
* The result is a flattened binary tree with right child pointers set correctly.
* **Time Complexity: O(N) as it visits each node exactly once.**
* **Space Complexity: O(H), where H is the height of the binary tree. In the worst case, where the tree is skewed, H could be N, making the space complexity O(N). In a balanced tree, it is O(log N).**

**Approach 2: Flatten a binary tree using an iterative approach (stack-based)**

* A public function **flattenIterative** is used to initiate the iterative flattening process.
* It uses a stack-based iterative approach.
* A stack is employed to traverse the binary tree in a pre-order manner, pushing nodes as they are visited.
* Right child pointers are correctly set as nodes are popped from the stack.
* The result is a flattened binary tree with right child pointers set correctly.
* **Time Complexity: O(N) as it visits each node exactly once.**
* **Space Complexity: O(H), where H is the height of the binary tree. In the worst case, where the tree is skewed, H could be N, making the space complexity O(N). In a balanced tree, it is O(log N).**

**Approach 3: Flatten Binary Tree to Linked List using the Iterative Morris Traversal**

* A public function **flattenMorrisTraversal** is used to initiate the Morris traversal flattening process.
* It utilizes Morris traversal, which doesn't use additional data structures.
* The key idea is to connect the right child of the in-order predecessor to the current node's right child and move to the left child.
* The left child of each node is set to null during the process.
* The result is a flattened binary tree with right child pointers set correctly.
* **Time Complexity: O(N) as it visits each node exactly once.**
* **Space Complexity: O(1) as it doesn't use additional data structures.**

**Conclusion:**

* All three approaches successfully flatten the binary tree into a linked list, maintaining the correct order of elements.
* The recursive approach is straightforward but uses additional space for the function call stack.
* The iterative approach eliminates the need for the function call stack and has a space complexity of O(H).
* The Morris traversal approach is the most memory-efficient with a space complexity of O(1).