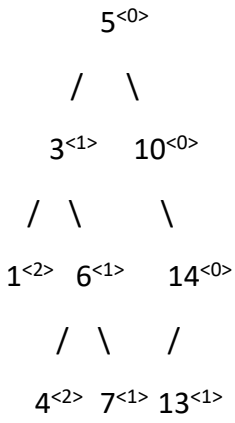
Diagonal Traversal of Binary Tree GFG

Given a Binary Tree, print the **diagonal traversal** of the binary tree.

Consider lines of slope -1 passing between nodes. Given a Binary Tree, print all diagonal elements in a binary tree belonging to same line.
If the diagonal element are present in two different subtress then left subtree diagonal element should be taken first and then right subtree.

Example:



Output: The Diagonal Traversal of Binary Tree: 5, 10, 14, 3, 6, 7, 13, 1, 4

Approach 1: Function to perform diagonal traversal of a binary tree using recursion.

- The recursive approach involves a helper function **solve** that traverses the tree diagonally.
- It maintains a map where each diagonal level is associated with a vector containing node values.
- The helper function adds the current node's value to the corresponding diagonal level.
- It recursively traverses the left subtree with an increased diagonal level and the right subtree with the same diagonal level.
- The main function initializes the map with the root node's value and then calls the **solve** function.
- Finally, it flattens the map and returns the values in the correct order.

Time Complexity:

- **The time complexity is O(N), where N is the number of nodes in the tree, as each node is visited once.**

Space Complexity:

- **The space complexity is O(H), where H is the height of the binary tree, due to the function call stack.**

Approach 2: Function to perform diagonal traversal of a binary tree iteratively.

- The iterative approach uses a queue for level-order traversal.
- It simulates diagonal traversal by using a while loop to traverse nodes on the same diagonal.
- The queue initially contains the root node.
- For each node popped from the queue, it traverses the diagonal by going left and enqueues the left child if it exists.
- It adds the node's value to the result vector.
- It then moves to the right child, simulating a diagonal movement.
- The process continues until the queue is empty.

Time Complexity:

- **The time complexity is O(N), where N is the number of nodes in the tree, as each node is visited once during level-order traversal.**

Space Complexity:

- **The space complexity is O(N), where N is the number of nodes in the tree, due to the queue.**

Conclusion:

Both the recursive and iterative approaches effectively perform diagonal traversal of a binary tree. The recursive approach uses depth-first traversal, while the iterative approach simulates diagonal traversal using level-order traversal. Both approaches have a time complexity of O(N) and provide the expected diagonal traversal sequence.

The choice between the two approaches depends on the specific requirements and constraints of the problem.