Problem Overview:

- The task is to return the values of nodes visible from the right side of a binary tree when given its root. We are required to simulate standing on the right side of the tree and listing the nodes from top to bottom. The problem involves traversing the tree and collecting the rightmost node at each level.

Problem Breakdown:

1. **Input:**

- ***The input is the root of a binary tree where:***
  - ➢ Each node has a value, a left child, and a right child.
  - ➢ The number of nodes is between 0 and 100.
  - ➢ The value of each node is between -100 and 100.

2. **Output:**

- The output is a list of integers that represent the values of nodes visible from the right side of the tree, ordered from the root to the deepest node.

Key Concepts:

1. **Binary Tree:**

- A data structure where each node can have at most two children: a left child and a right child. The topmost node is referred to as the root, and each child can itself be the root of a subtree.

2. **Right Side View:**

- Imagine looking at the tree from the right side. At each level, only the rightmost node is visible. Our goal is to collect these visible nodes.

3. Level-Order Traversal:

- A method of tree traversal where nodes are visited level by level. In this approach, we visit all nodes on the same level before moving to the next level. It is typically implemented using a queue data structure.

4. Queue:

- A data structure that operates in a first-in-first-out (FIFO) manner. It is used in level-order traversal to keep track of nodes that need to be processed.

Approach to Solve:

1. Initial Condition:

- If the tree is empty (i.e., root is None), return an empty list immediately because no nodes are visible from any side.

2. Level-Order Traversal Using a Queue:

- Initialize a queue and add the root node to it.
- Use the queue to process each level of the tree one by one.
- ***For each level:***
  - ➢ Determine how many nodes are on that level (this is the size of the queue at the start of the level).
  - ➢ Traverse each node in the current level.
  - ➢ *For every node:*
    - ✓ If it has a left child, add it to the queue.
    - ✓ If it has a right child, add it to the queue.
  - ➢ Once all nodes on the level are processed, append the value of the last node (rightmost) to the result list, as that is the node visible from the right side.

3. **Result Collection:**

- For every level of the tree, collect the value of the rightmost node and store it in a result list.
- Continue this process until all levels are processed and the queue is empty.

4. **Edge Cases:**

- ***Empty Tree:*** When the input tree is empty, the output should be an empty list.
- ***Single Node Tree:*** If the tree has only one node, the right side view will only contain that node's value.
- ***All Left Nodes or All Right Nodes:*** The method still works in these cases since at each level, only the rightmost node (whether it's on the left or right subtree) will be considered.

Time Complexity:

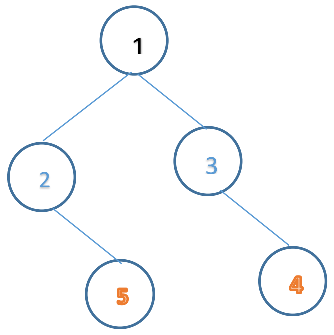
- **Traversal Time:** Since we visit each node exactly once during the level-order traversal, the time complexity is proportional to the number of nodes, i.e., O(n), where n is the total number of nodes in the binary tree.

Space Complexity:

- **Queue Size:** At most, the queue will contain all nodes of the largest level of the tree. In the worst case, for a completely balanced binary tree, the last level contains approximately n/2 nodes. Therefore, the space complexity of the queue is O(n).

Example 1:

Input: A binary tree with the following structure:

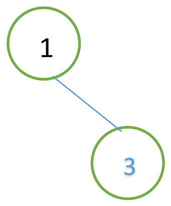


Output: [1, 3, 4]

Explanation: From the right side, the visible nodes are 1, 3, and 4.

2. Example 2:

Input: A binary tree with the following structure:



Output: [1, 3]

Explanation: From the right side, the visible nodes are 1 and 3.

Example 3:

Input: An empty tree (root = []).

Output: []

Explanation: There are no nodes in the tree, so nothing is visible from the right side.

Constraints:

- **Tree Size:** The number of nodes in the tree will be in the range [0, 100], which means we are dealing with a small binary tree, so an O(n) solution is efficient.

- **Node Values:** Each node will have a value between -100 and 100. This means we must ensure that the tree can handle negative numbers.

Final Remarks:

- The problem is a straightforward application of a level-order traversal (BFS), combined with the concept of collecting only the rightmost nodes. It tests the ability to navigate through a binary tree in a systematic manner, using queues and considering edge cases like empty trees or skewed trees.