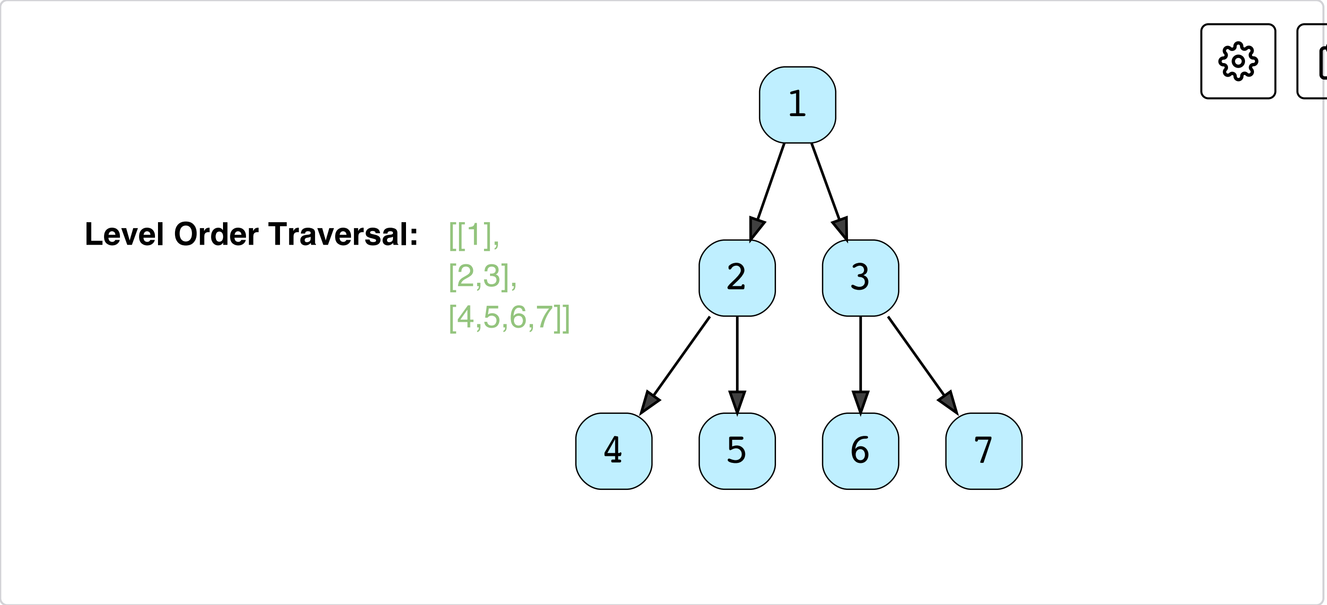
Binary Tree Level Order Traversal (easy)

We'll cover the following

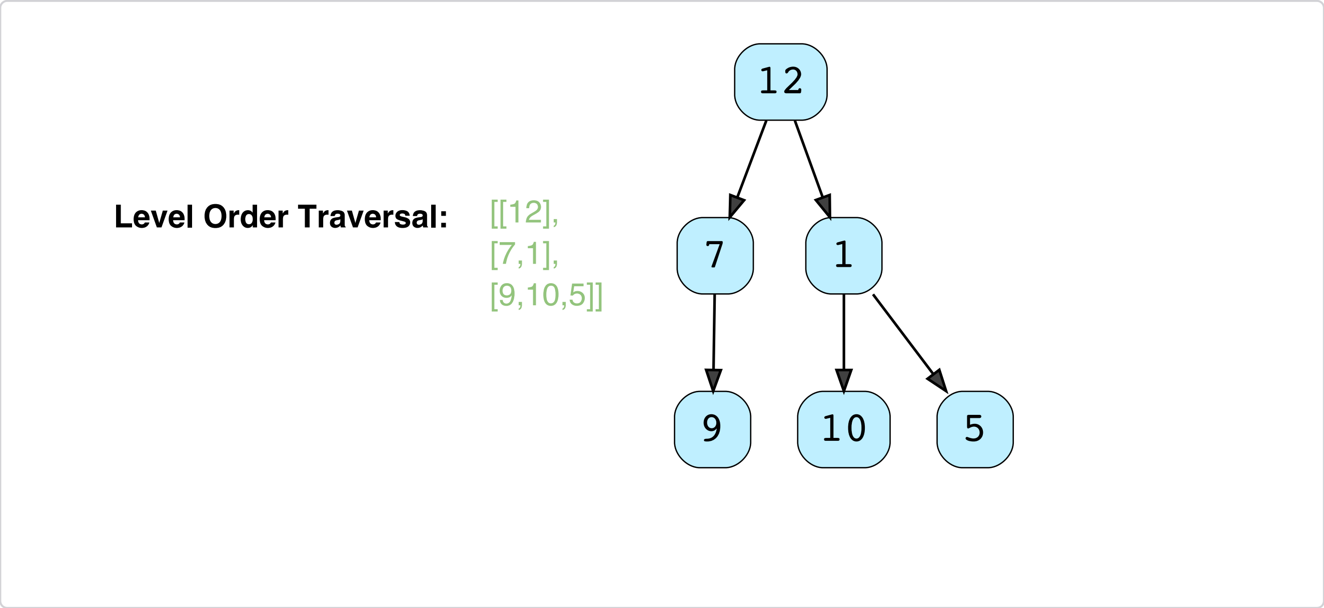
Problem Statement

Given a binary tree, populate an array to represent its level-by-level traversal. You should populate the values of all **nodes of each level from left to right** in separate sub-arrays.

Example 1:



Example 2:

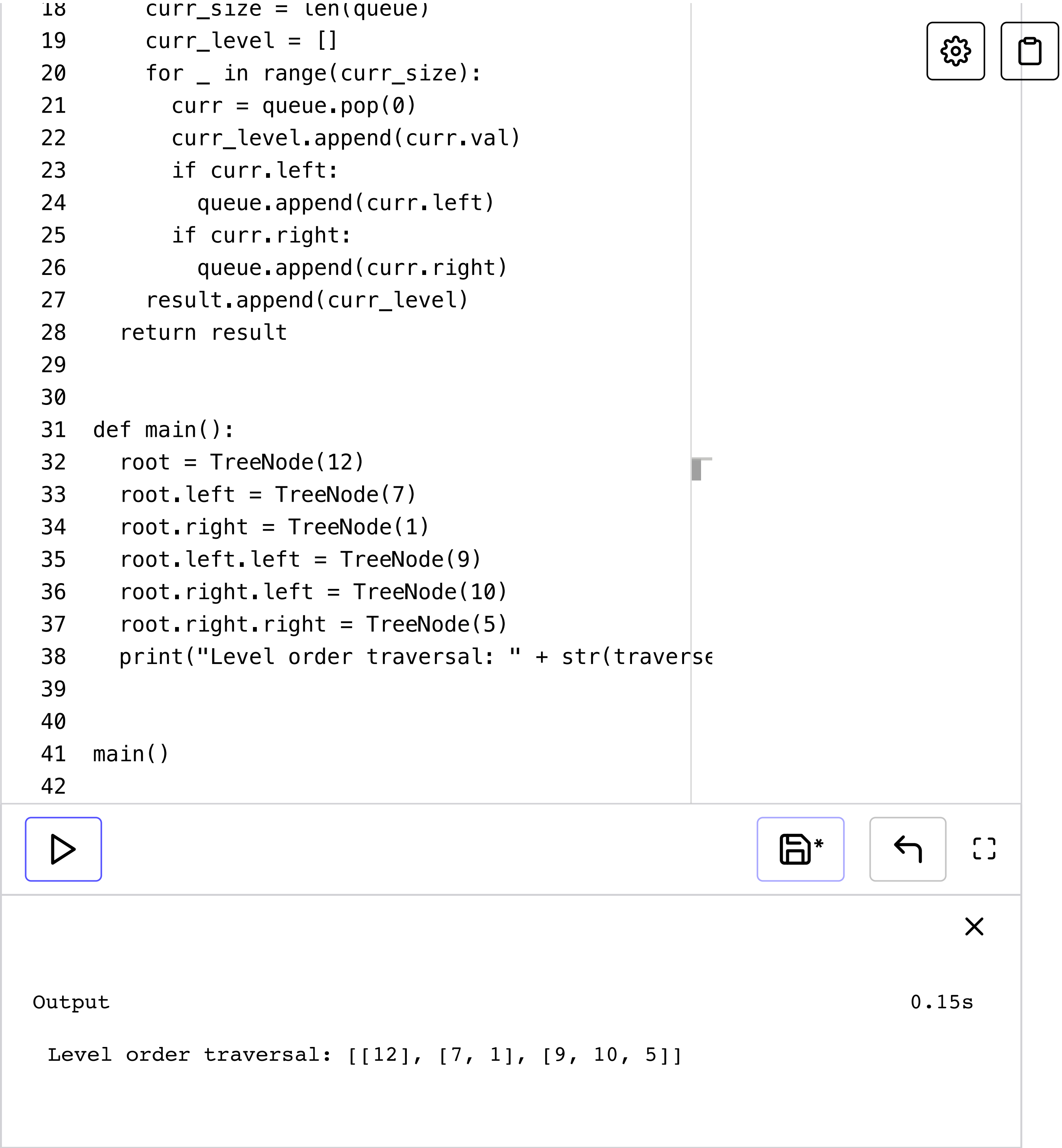


Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
    queue = [root,]
    # when queue not empty
    while queue:
```



```
    curr_size = len(queue)
    curr_level = []
    for _ in range(curr_size):
      curr = queue.pop(0)
      curr_level.append(curr.val)
      if curr.left:
        queue.append(curr.left)
      if curr.right:
        queue.append(curr.right)
    result.append(curr_level)
  return result


def main():
  root = TreeNode(12)
  root.left = TreeNode(7)
  root.right = TreeNode(1)
  root.left.left = TreeNode(9)
  root.right.left = TreeNode(10)
  root.right.right = TreeNode(5)
  print("Level order traversal: " + str(traverse


main()

```

```
Output                                                    0.15s

 Level order traversal: [[12], [7, 1], [9, 10, 5]]
```

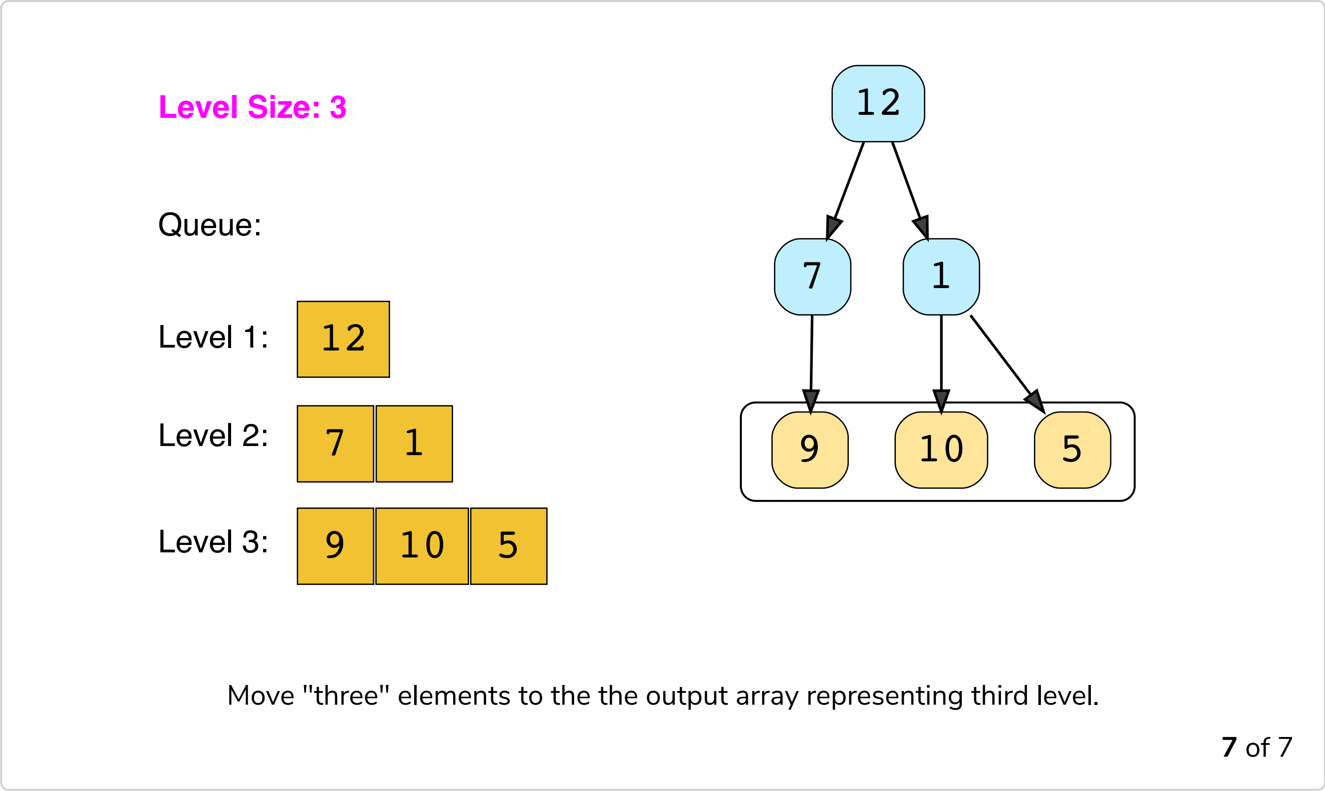
Solution

Since we need to traverse all nodes of each level before moving onto the next level, we can use the **Breadth First Search (BFS)** technique to solve this problem.

We can use a Queue to efficiently traverse in BFS fashion. Here are the steps of our algorithm:

1. Start by pushing the `root` node to the queue.
2. Keep iterating until the queue is empty.
3. In each iteration, first count the elements in the queue (let's call it `levelSize`). We will have these many nodes in the current level.
4. Next, remove `levelSize` nodes from the queue and push their `value` in an array to represent the current level.
5. After removing each node from the queue, insert both of its children into the queue.
6. If the queue is not empty, repeat from step 3 for the next level.

Let's take the example-2 mentioned above to visually represent our algorithm:



Level Size: 3

Queue:

Level 1: 12

Level 2: 7 1

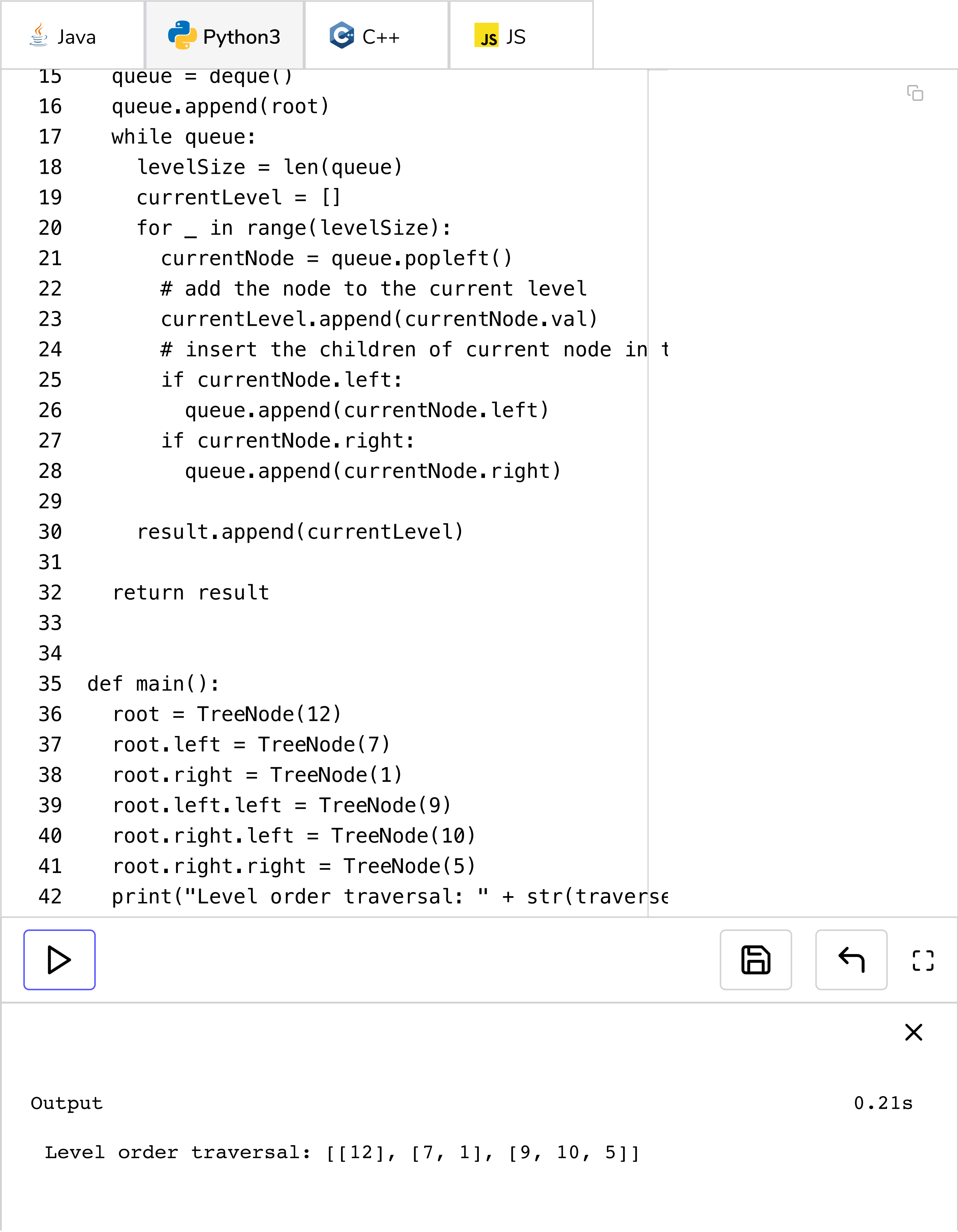
Level 3: 9 10 5

Move "three" elements to the the output array representing third level.

7 of 7

Code

Here is what our algorithm will look like:



```python
  queue = deque()
  queue.append(root)
  while queue:
    levelSize = len(queue)
    currentLevel = []
    for _ in range(levelSize):
      currentNode = queue.popleft()
      # add the node to the current level
      currentLevel.append(currentNode.val)
      # insert the children of current node in t
      if currentNode.left:
        queue.append(currentNode.left)
      if currentNode.right:
        queue.append(currentNode.right)

    result.append(currentLevel)

  return result


def main():
  root = TreeNode(12)
  root.left = TreeNode(7)
  root.right = TreeNode(1)
  root.left.left = TreeNode(9)
  root.right.left = TreeNode(10)
  root.right.right = TreeNode(5)
  print("Level order traversal: " + str(traverse
```

```
Output                                                        0.21s

 Level order traversal: [[12], [7, 1], [9, 10, 5]]
```

Time complexity

The time complexity of the above algorithm is $O(N)$, where ‘N’ is the total number of nodes in the tree. This is due to the fact that we traverse each node once.

Space complexity

The space complexity of the above algorithm will be $O(N)$ as we need to return a list containing the level order traversal. We will also need $O(N)$ space for the queue. Since we can have a maximum of $N/2$ nodes at any level (this could happen only at the lowest level), therefore we will need $O(N)$ space to store them in the queue.