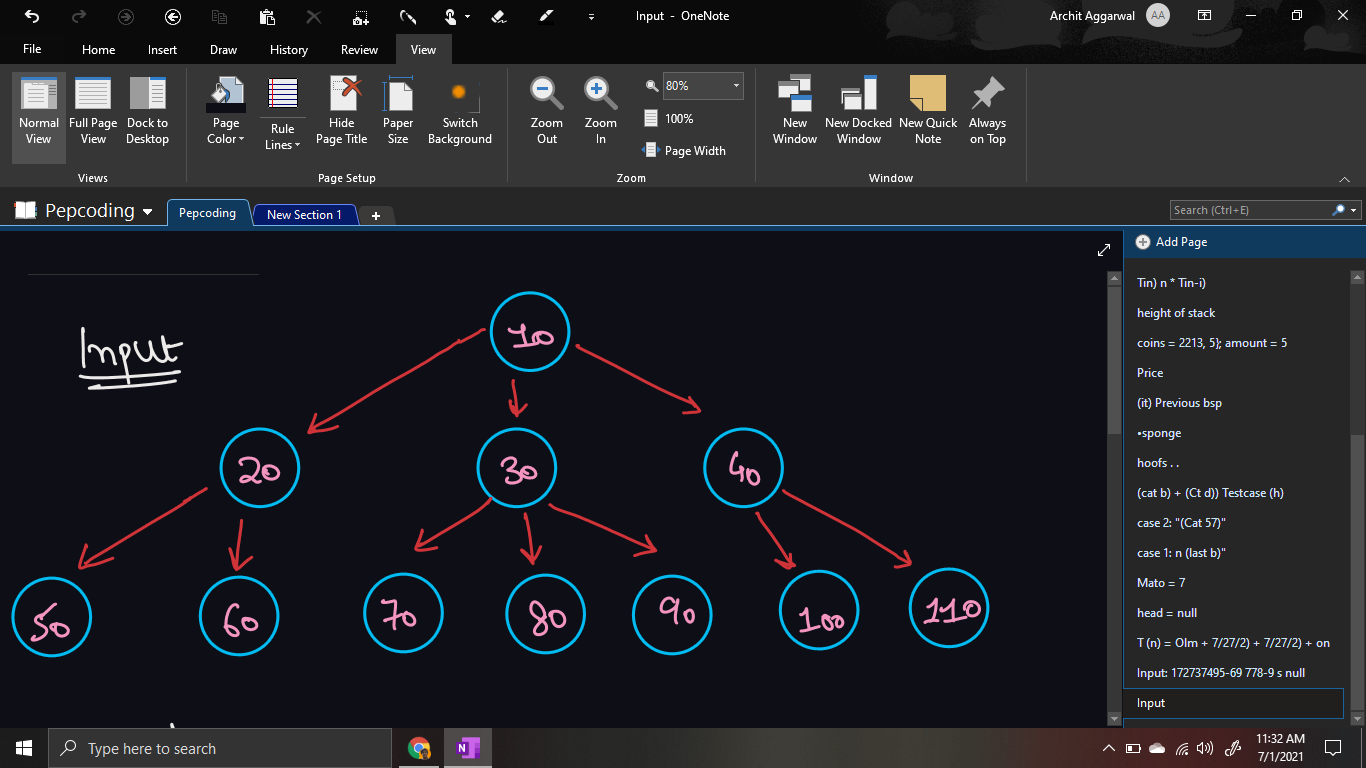
Dear reader, welcome to the article on the topic named **‘Level Order Traversal - More Approaches’.**

Moving further, this article has a prerequisite that you must know how to do ‘[**level order traversal of a generic tree**](https://www.pepcoding.com/resources/online-java-foundation/generic-tree/level-order-generic-tree-official/ojquestion)’ and ‘[**level order traversal linewise**](https://www.pepcoding.com/resources/online-java-foundation/generic-tree/levelorder-linewise-generic-tree-official/ojquestion)’ using two queues. You can try to submit these approaches in the ‘[level order traversal linewise](https://www.pepcoding.com/resources/online-java-foundation/generic-tree/levelorder-linewise-generic-tree-official/ojquestion)’ question only.

***Problem Statement:***

* You are given a partially written GenericTree class. You are required to complete the body of levelorderLineWise function. (Input is managed for you).
* The function is expected to visit every node in "level order fashion" and print them from left to right (level by level).
* All nodes on the same level should be separated by a space. Different levels should be on separate lines.

Please check the [question video](https://www.youtube.com/watch?v=t4ts_6m4z68&list=TLGGrgxh0VAiqj8wMTA3MjAyMQ) for more details. For example for the following generic tree:



The output should be:

*10*

*20 30 40*

*50 60 70 80 80 90 100 110*

***Solution 1: Using Two Queues:***

This solution, which uses two queues, (main queue and child queue), has been discussed already in this [article](https://drive.google.com/file/d/1IdwN8JkHL2ueTnwI4NT23-TaASRwwovb/view).

We will solve this problem using one queue only in the remaining approaches.

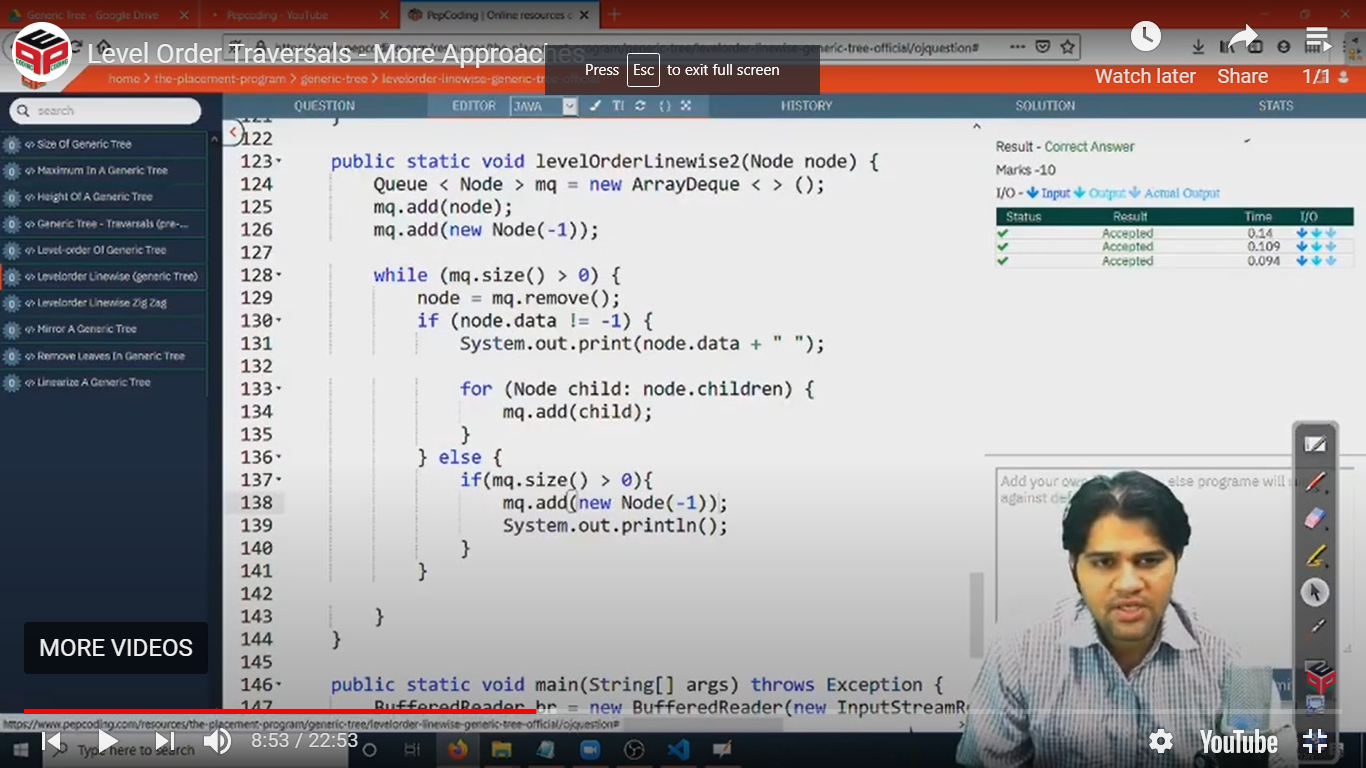
***Solution 2: Using Dummy Node:***

***Idea***: We can mark the end of the current level by inserting a dummy node (with value -1), which can help us know that the next node to be popped will be in the next level.

* Initially, we will insert the root node and a dummy node as well before the while loop.
* Now, we will run the while loop until the queue becomes empty.
  + Dequeue the front element of the queue. Print the node value, and now, there will be two cases:
    - If the current element is not a dummy node (value is not -1), then simply enqueue all the children of the node in the queue.
    - Else, the dummy node marks the end of the current level. Hence, we should print a new line, and also enqueue another dummy node at the end of the queue (to mark the end of the next level).
    - But, before printing a new line and adding another dummy node, just check whether the queue is empty or not. If the queue is empty, we just need to break as the current level is the last one and there are no more levels to be processed.

***Important Note***: If we push another dummy node for the last level also, then we will get stuck in an infinite loop, as we will keep on popping one and adding another dummy node, and the size of the queue will never become 0.

***Implementation:***



***Solution 3: Using Size Variable:***

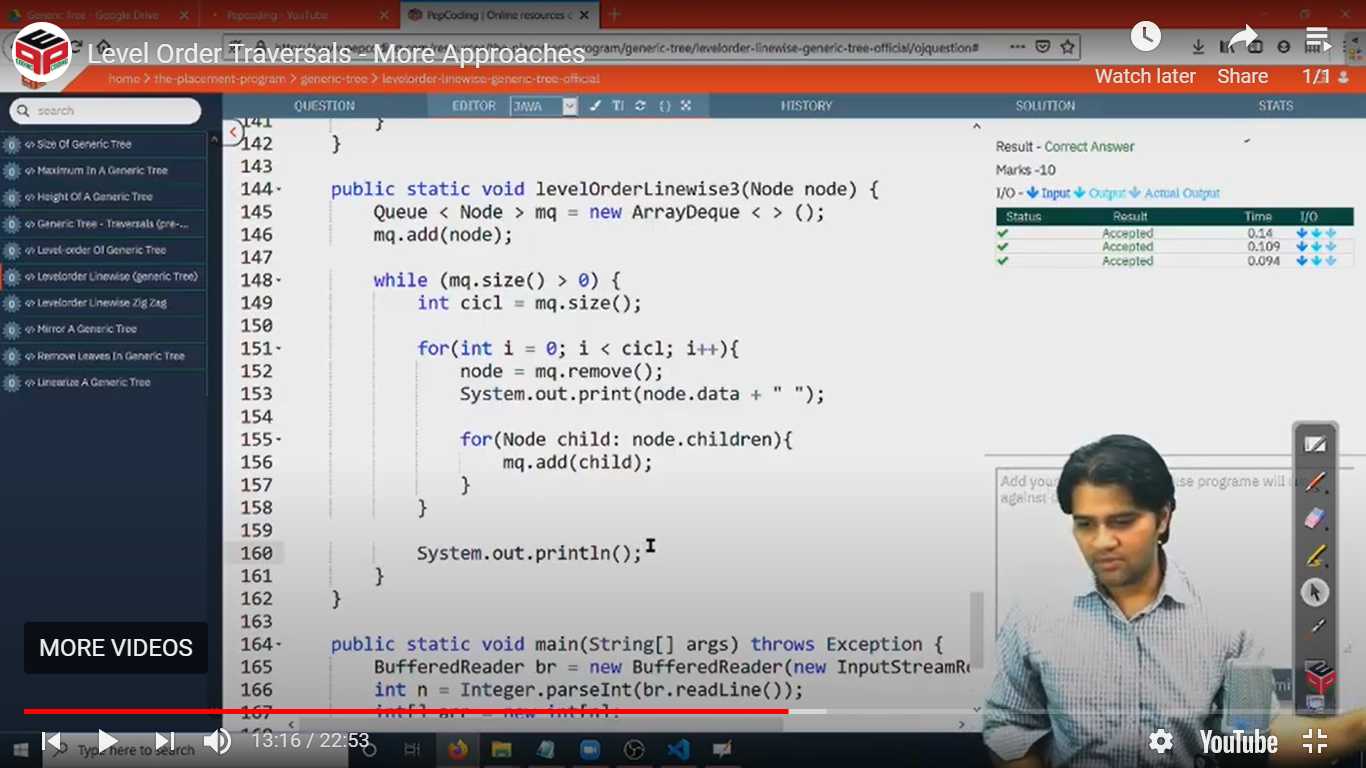
***Idea***: We can store the size of the current level in an integer variable. Then, we can run a loop equal to size times and push the children nodes (next level) in the queue.

By storing the size in another variable, we will not have to worry about marking the pointer of the end of the current level, as we will run a loop for size number of times.

* Initially, we will insert the root node in the queue before the while loop.
* Now, we will run the while loop until the queue becomes empty.
  + Store the size of queue (number of nodes in current level) in an integer variable *size*.
  + Run a loop for size number of times.
    - Dequeue one element from the front of the queue, print the node value and enqueue all of it’s children nodes in the queue.
  + Print a newline to output the next level in the next line.

***Important Note***: We have stored size and not used mq.size() directly in the for loop, as it will keep on increasing after adding the child nodes, and eventually all nodes will get printed in the same level.

***Implementation:***



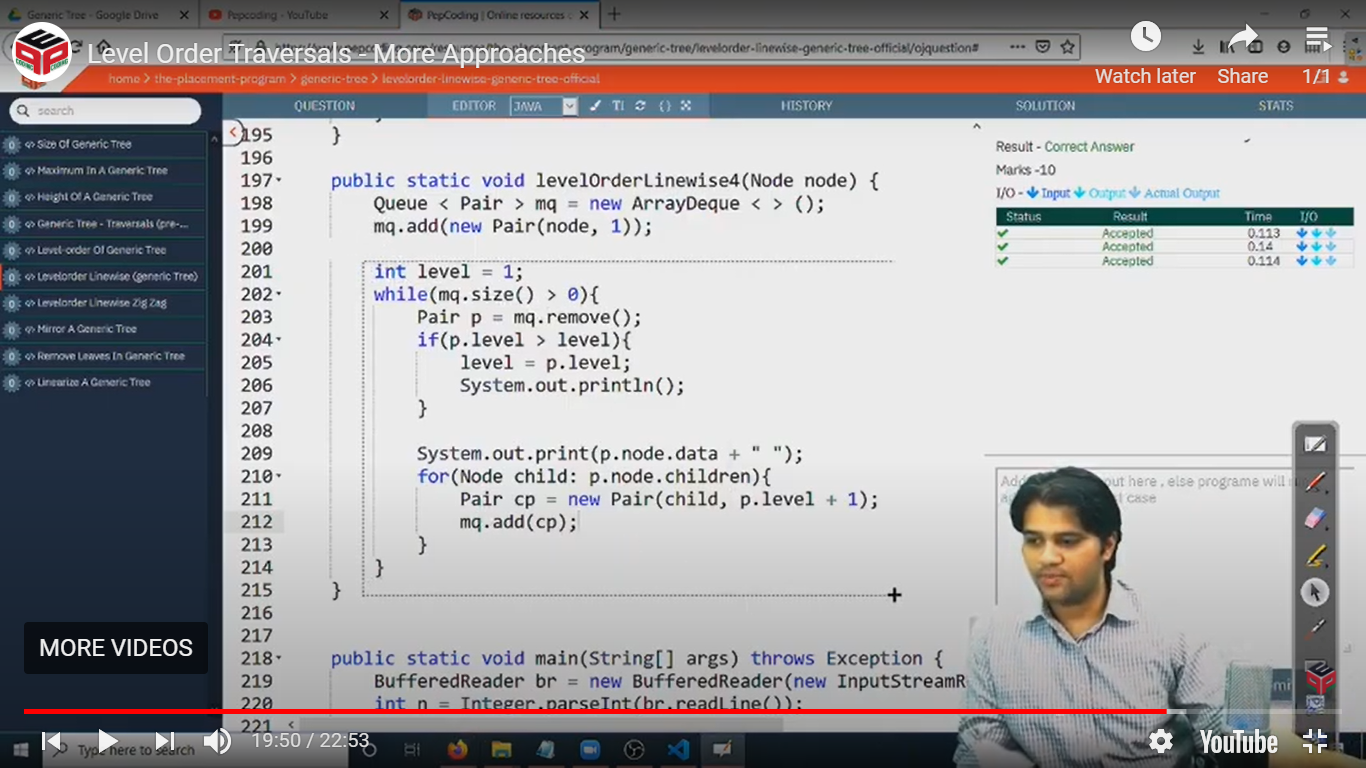
***Solution 4: Using Pair Node:***

***Idea***: Instead of just pushing the node’s value, we can also push it along with its level number, i.e. make a ***pair object of node and level***. Whenever we encounter a node with a level greater than the level of the previous node, we will print a new line also after printing the node’s value.

* Initially, we will insert the root node with it’s level as 1.
* We will initialize the current level in an integer variable as 1.
* Now, we will run the while loop until the queue becomes empty.
  + Dequeue the front element of the queue. Print the node value. Push all the children of the nodes with level 1 more than the node’s level.
  + There will be two cases:
    - If the node’s level is equal to the current level, then do not do anything else.
    - If the node’s level is greater than the current level, then also print a new line and update the current level as the node's level.

***Important Note***: This method takes extra memory for storing levels of the node. But the space complexity will remain the same, and that is O(n).

***Implementation:***



* What is the ***time complexity*** of the above codes?

All the approaches will have equivalent time complexity, as we will be pushing/popping each node exactly once in all the approaches. Thus, the time complexity will be ***O(n)*** where n = number of nodes in a generic tree.

* What is the ***space complexity*** of the above codes?

We are using an auxiliary ***queue data structure*** for level order traversal. At any time, the maximum size of the queue will be equal to the maximum nodes in any level of the generic tree. Thus, ***O(n) auxiliary space*** is used.

Hope that you liked the article on *Level Order Traversal - More Approaches*.

***Suggestion***: Please try to keep in mind atleast one of the approach discussed today. It is recommend to know the ***3rd approach (using size variable)***, as it does not store any dummy node, and also it does not need to maintain levels of the node.

Subscribe to Pepcoding’s youtube channel for more such amazing content on Data Structures & Algorithms and follow the resources available for all students in the resources section of Pepcoding’s website.

You can suggest any improvements to the article on our telegram channel, or on the youtube channel’s comment section.

Article Contributed by: [Archit Aggarwal](https://www.linkedin.com/in/archit-aggarwal-6a7716189/)