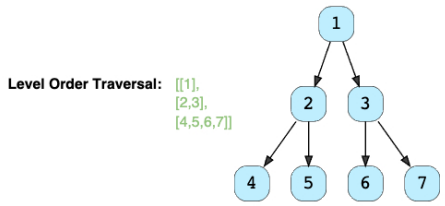
Binary Tree Level Order Traversal (easy)

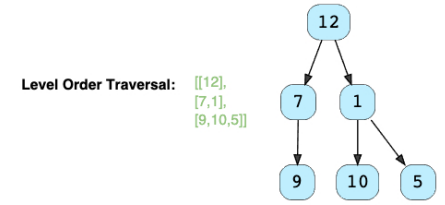
Problem Statement

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++

```
class TreeNode {

  constructor(value) {
    this.value = value;
    this.left = null;
    this.right = null;
  }
};


const traverse = function(root) {
  result = [];
  // TODO: Write your code here
  return result;
};



var root = new TreeNode(12);
root.left = new TreeNode(7);
root.right = new TreeNode(1);
root.left.left = new TreeNode(9);
root.right.left = new TreeNode(10);
root.right.right = new TreeNode(5);
console.log(`Level order traversal: ${traverse(root)}`);

```

RUN SAVE RESET

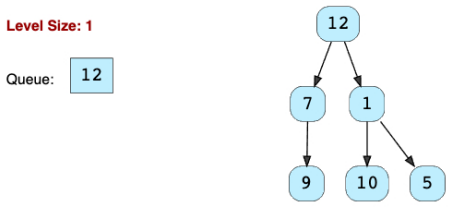
Solution

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:

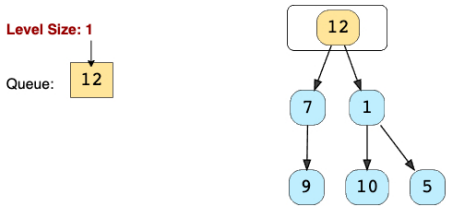
1. Start by pushing the `root` node to the queue.
2. Keep iterating until the queue is empty.
3. In each iteration, first count the elements in the queue (let's call it `levelSize`). We will have these many nodes in the current level.
4. Next, remove `levelSize` nodes from the queue and push their `value` 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:



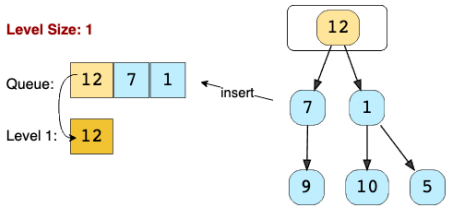
Start by pushing the root to the queue.

1 of 7



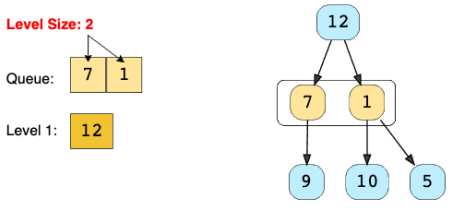
Count the elements of the queue (levelSize = 1), they all will be in the first level. Since the levelSize is "1" there will be one element in the first level.

2 of 7



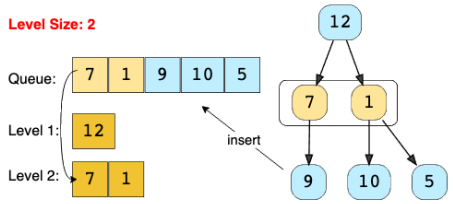
Move "one" element to the the output array representing the first level and push its children to the queue.

3 of 7



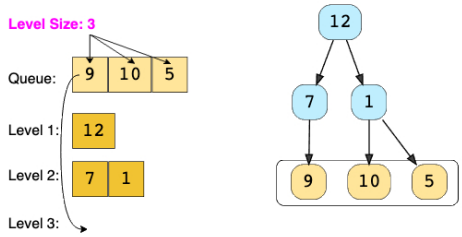
Count the elements of the queue (levelSize = 2), they all will be in the second level. Since the levelSize is "2" there will be two elements in the second level.

4 of 7



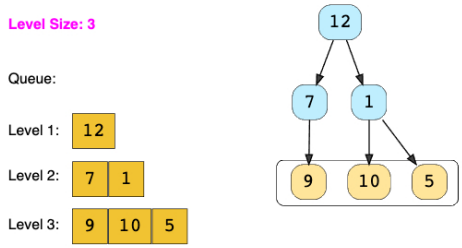
Move "two" elements to the the output array representing the second level and push their children to the queue in the same order.

5 of 7



Count the elements of the queue (levelSize = 3), they all will be in the third level. Since the levelSize is "3" there will be three elements in the third level.

6 of 7



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

7 of 7

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
const Deque = require('./collections/deque'); //http://www.collectionsjs.com


class TreeNode {
  constructor(val) {
    this.val = val;
    this.left = null;
    this.right = null;
  }
}


function traverse(root) {
  result = [];
  if (root === null) {
    return result;
  }

  const queue = new Deque();
  queue.push(root);
  while (queue.length > 0) {
    const levelSize = queue.length;
    currentLevel = [];
    for (i = 0; i < levelSize; i++) {
      currentNode = queue.shift();
      // add the node to the current level
      currentLevel.push(currentNode.val);
      // insert the children of current node in the queue
```

RUN SAVE RESET

Close

Mark Course as Completed

Output

```
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.