

## Grokking the Coding Interview: Patterns for Coding Questions

30% completed

Search Course

Solution Review: Problem Challenge 3

### Pattern: Merge Intervals

- Introduction
- Merge Intervals (medium)
- Insert Interval (medium)
- Intervals Intersection (medium)
- Conflicting Appointments (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

### Pattern: Cyclic Sort

- Introduction
- Cyclic Sort (easy)
- Find the Missing Number (easy)
- Find all Missing Numbers (easy)
- Find the Duplicate Number (easy)
- Find all Duplicate Numbers (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

### Pattern: In-place Reversal of a LinkedList

- Introduction
- Reverse a LinkedList (easy)
- Reverse a Sub-list (medium)
- Reverse every K-element Sub-list (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

### Pattern: Tree Breadth First Search

- Introduction
- Binary Tree Level Order Traversal (easy)
- Reverse Level Order Traversal (easy)
- Zigzag Traversal (medium)

## Binary Tree Level Order Traversal (easy)

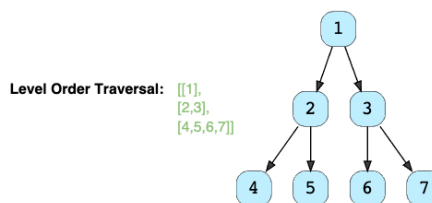
We'll cover the following

- Problem Statement
- Try it yourself
- Solution
- Code
  - Time complexity
  - Space complexity

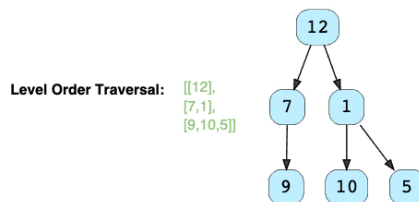
### Problem Statement

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:

```
1 class TreeNode {
2
3   constructor(value) {
4     this.value = value;
5     this.left = null;
6     this.right = null;
7   }
8 };
9
10
11 const traverse = function(root) {
12   result = [];
13   // TODO: Write your code here
14   return result;
15 };
16
17
18
19 var root = new TreeNode(12);
20 root.left = new TreeNode(7);
21 root.right = new TreeNode(1);
22 root.left.left = new TreeNode(9);
23 root.right.left = new TreeNode(10);
24 root.right.right = new TreeNode(5);
25 console.log('Level order traversal: ' + traverse(root));
26
```

RUN

SAVE

RESET

🔄

- Level Averages in a Binary Tree (easy)
- Minimum Depth of a Binary Tree (easy)
- Level Order Successor (easy)
- Connect Level Order Siblings (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

## Pattern: Tree Depth First Search

- Introduction
- Binary Tree Path Sum (easy)
- All Paths for a Sum (medium)
- Sum of Path Numbers (medium)
- Path With Given Sequence (medium)
- Count Paths for a Sum (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

## Pattern: Two Heaps

- Introduction
- Find the Median of a Number Stream (medium)
- Sliding Window Median (hard)
- Maximize Capital (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

## Pattern: Subsets

- Introduction
- Subsets (easy)
- Subsets With Duplicates (easy)
- Permutations (medium)
- String Permutations by changing case (medium)
- Balanced Parentheses (hard)
- Unique Generalized Abbreviations (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

## Pattern: Modified Binary Search

- Introduction
- Order-agnostic Binary Search (easy)
- Ceiling of a Number (medium)
- Next Letter (medium)
- Number Range (medium)
- Search in a Sorted Infinite Array (medium)
- Minimum Difference Element (medium)
- Bitonic Array Maximum (easy)

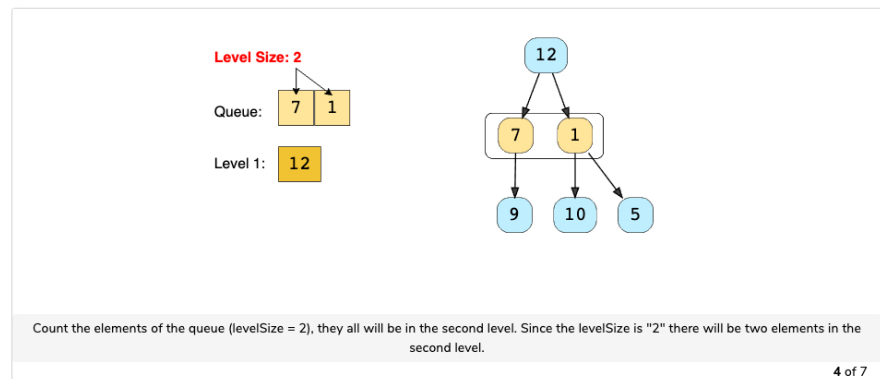
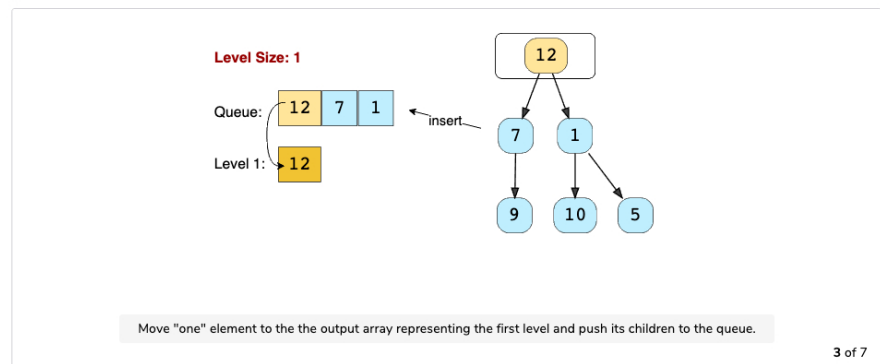
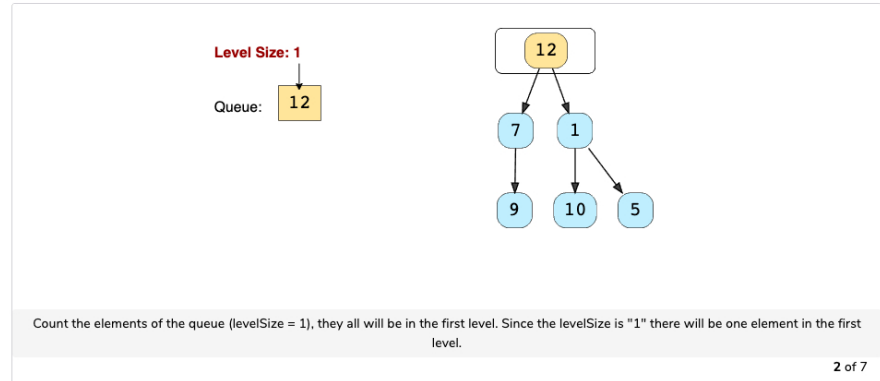
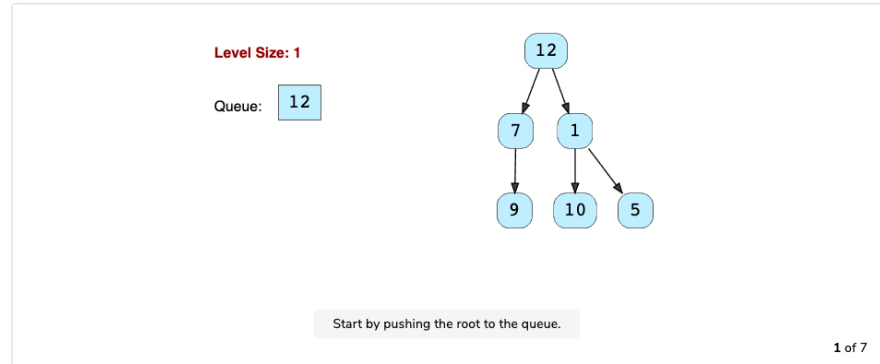
## Solution

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:



- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

## Pattern: Bitwise XOR

- Introduction
- Single Number (easy)
- Two Single Numbers (medium)
- Complement of Base 10 Number (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

## Pattern: Top 'K' Elements

- Introduction
- Top 'K' Numbers (easy)
- Kth Smallest Number (easy)
- 'K' Closest Points to the Origin (easy)
- Connect Ropes (easy)
- Top 'K' Frequent Numbers (medium)
- Frequency Sort (medium)
- Kth Largest Number in a Stream (medium)
- 'K' Closest Numbers (medium)
- Maximum Distinct Elements (medium)
- Sum of Elements (medium)
- Rearrange String (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

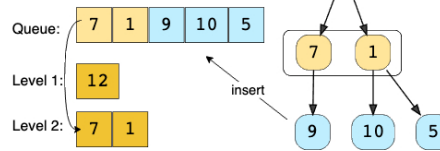
## Pattern: K-way merge

- Introduction
- Merge K Sorted Lists (medium)
- Kth Smallest Number in M Sorted Lists (Medium)
- Kth Smallest Number in a Sorted Matrix (Hard)
- Smallest Number Range (Hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

## Pattern : 0/1 Knapsack (Dynamic Programming)

- Introduction
- 0/1 Knapsack (medium)
- Equal Subset Sum Partition (medium)
- Subset Sum (medium)
- Minimum Subset Sum Difference (hard)
- Problem Challenge 1
- Solution Review: Problem

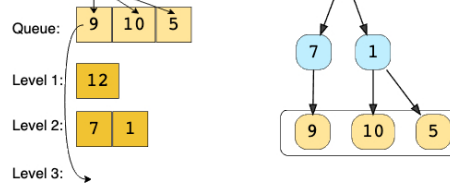
Level Size: 2



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

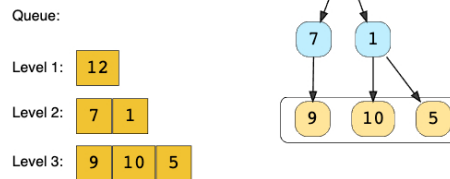
Level Size: 3



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

Level Size: 3



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

```

1  const Deque = require('./collections/deque'); //http://www.collectionsjs.com
2
3
4  class TreeNode {
5      constructor(val) {
6          this.val = val;
7          this.left = null;
8          this.right = null;
9      }
10 }
11
12
13 function traverse(root) {
14     result = [];
15     if (root === null) {
16         return result;
17     }
18
19     const queue = new Deque();
20     queue.push(root);
21     while (queue.length > 0) {
22         const levelSize = queue.length;
23         currentLevel = [];
24         for (i = 0; i < levelSize; i++) {
25             currentNode = queue.shift();
26             // add the node to the current level
27             currentLevel.push(currentNode.val);
28             // insert the children of current node in the queue

```

RUN

SAVE

RESET

Close

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Pattern: Topological Sort (Graph)

Introduction

Topological Sort (medium)

Tasks Scheduling (medium)

Tasks Scheduling Order (medium)

All Tasks Scheduling Orders (hard)

Alien Dictionary (hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Miscellaneous

Kth Smallest Number (hard)

Conclusions

Where to Go from Here

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.

Interviewing soon? We've partnered with Hired so that companies apply to you, instead of the other way around. [See how](#)

← Back

Introduction

Reverse Level Order Traversal (easy)

MARK AS COMPLETED

Next →

Report an Issue