TIP101 | Intro to Technical Interview Prep

Intro to Technical Interview Prep Fall 2025 (@ Section 3 | Tuesdays and Thursdays 5PM - 7PM PT)
Personal Member ID#: **134071**

Need help? Post on our [class slack channel]() or email us at [support@codepath.org]()

Getting Started

Learning with AI ✨

IDE Setup

HackerRank Guide

Schedule

Course Progress

Unit 1

Unit 2

Unit 3

Unit 4

Unit 5

Unit 6

Unit 7

Unit 8

Unit 9

Unit 10

nission Guide

Session 2: Binary Trees Essentials

Overview

In this session, students will learn various techniques related to binary tree traversal and depth calculations. The curriculum will cover tasks such as performing level order traversal, finding the minimum depth, and calculating odd-even level sums. These exercises will help students understand breadth-first search (BFS) and its applications in binary tree problems.

Students will develop skills in analyzing tree structures and calculating specific properties, preparing them for more complex tree-related problems.

> You can find all resources from today including the session deck, session recording, and more on the resources tab

🎢 Part 1 : Instructor Lead Session

We'll spend the first portion of the synchronous class time in large groups, where the instructor will lead class instruction for 30-45 minutes.

🧑‍💻 Part 2: Breakout Session

In breakout sessions, we will explore and collaboratively solve problem sets in small groups. Here, the **collaboration, conversation, and approach** are just as important as "solving the problem" - please engage warmly, clearly, and plentifully in the process!

In breakout rooms you will:

- Screen-share the problem/s, and verbally review them together
- Screen-share an interactive coding environment, and talk through the steps of a solution approach
  - ProTip: - An Integrated Development Environment (IDE) is a fancy name for a tool you could use for shared writing of code - like Replit.com, Collabed.it, CodePen.io, or other - your staff team will specify which tool to use for this class!
- Screen-share an implementation of your proposed solution
- Independently follow-along, or create an implementation, in your own IDE.

ur program leader/s will indicate which code sharing tool/s to use as a group, and will help break down provide specific scaffolding with the main concepts above.

🔎 Problem Solving Approach

We will approach problems using the six steps in the UMPIRE approach.

UMPIRE: Understand, Match, Plan, Implement, Review, Evaluate.

We'll apply these six steps to the problems we'll see in the first half of the course.

We will learn to:

- **Understand** the problem
- **Match** identifies common approaches you've seen/used before
- **Plan** a solution step-by-step, and
- **Implement** the solution
- **Review** your solution
- **Evaluate** your solution's time and space complexity and think critically about the advantages and disadvantages of your chosen approach.

Breakout Problems Session 2

💡 Unit 9 Cheatsheet

To help your learning journey with binary trees, we've put together a guide to common concepts and syntax you will use throughout Unit 9 breakout problems. Use this cheatsheet as a quick reference guide as you work through the problems below.

▾ Problem Set Version 1

Problem 1: Level Order Traversal of Binary Tree

Given the following pseudocode and the `root` of a binary tree, return a list of the level order traversal of it's nodes' values (i.e., from left to right, level by level).

Evaluate the time complexity of your solution. Define your variables and give a rationale as to why you believe your solution has the stated time complexity.

```
class TreeNode:
    def __init__(self, value=0, left=None, right=None):
        self.val = value
```

```python
def level_order(root):
    # If the tree is empty:
    # return an empty list

    # Create an empty queue using deque
    # Create an empty list to store the explored nodes

    # Add the root to the queue

    # While the queue is not empty:
    # Pop the next node off the queue (pop from the left side!)
    # Add the popped node to the list of explored nodes

    # Add each of the popped node's children to the end of the queue

    # Return the list of visited nodes
    pass
```

Example Usage:

```
Example Input Tree:

      4
     / \
    2   6
   / \
  1   3

Example Input: root = 4
Expected Output: [4, 2, 6, 1, 3]
Explanation:
Level 1: Node 4
Level 2 (left to right): Node 2, Node 6
Level 3 (left to right): Node 1, Node 3
```

▶ ✨ **AI Hint: Breadth First Search Traversal**

Problem 2: Find Minimum Depth of Binary Tree

Given the `root` of a binary tree, return its minimum depth. The minimum depth is the number of nodes along the shortest path from the `root` down to the nearest leaf node.

Evaluate the time complexity of your solution. Define your variables and give a rationale as to why you believe your solution has the stated time complexity.

```
        self.val = value
        self.left = left
        self.right = right


def min_depth(root):
        pass
```

Example Usage:

```
Example Input Tree #1:

   3
  / \
 9  20
    / \
   15  7

Example Input: root = 3
Expected Output: 2
Shortest path from root node to a leaf node is 3 -> 9. Number of nodes in path is 2.

Example Input Tree #2:

   2
    \
     3
      \
       4
        \
         5
          \
           6

Example Input: root = 2
Expected Output: 5
Shortest path from root node to a leaf node is 2 -> 3 -> 4 -> 5 -> 6.
Number of nodes in path is 5.
```

▶ 💡 **Hint: Choosing your Traversal Method**

Problem 3: Odd-Even Level Sum Difference in Binary Tree

Given the `root` of a binary tree, return the difference between the sum of all node values in odd levels and sum of all node values in even levels.

```python
class TreeNode:
    def __init__(self, value=0, left=None, right=None):
        self.val = value
        self.left = left
        self.right = right

def level_difference(root):
        pass
```

Example Usage:

```
Example Input Tree
          6
         / \
        3   8
       /   / \
      5   4   2
         / \   \
        1   7   3
Expected Output: -5
Explanation:
Odd level sum: 6 + 5 + 4 + 2 = 17
Even level sum: 3 + 8 + 1 + 7 + 3 = 22
Odd level sum - even level sum: 17 - 22 = -5
```

Problem 4: Level Order Traversal of Binary Tree with Nested Lists

Given the `root` of a binary tree, write a function `level_order()` that returns the level order traversal of its nodes' values (i.e., from left to right, level by level). `level_order()` should return a list of lists, where each inner list contains the node values of a single level in the tree.

Evaluate the time complexity of your solution. Define your variables and give a rationale as to why you believe your solution has the stated time complexity.

```python
class TreeNode:
    def __init__(self, value=0, left=None, right=None):
        self.val = value
        self.left = left
        self.right = right

def level_order(root):
        pass
```

```
     3
    / \
   9  20
      / \
     15  7

Input: root = 3
Expected Output: [ [3], [9, 20], [15, 7]]
```

▶ ✨ **AI Hint: Nested lists**

Problem 5: Sum of Binary Tree Node Tilts

Given the `root` of a binary tree, return the sum of every tree node's tilt. The tilt of a tree node is the absolute difference between the sum of all left subtree node values and all right subtree node values. If a node does not have a left child, then the sum of the left subtree node values is treated as `0`. The rule is similar if the node does not have a right child.

Evaluate the time complexity of your solution. Define your variables and give a rationale as to why you believe your solution has the stated time complexity.

```python
class TreeNode:
    def __init__(self, value=0, left=None, right=None):
        self.val = value
        self.left = left
        self.right = right



def find_tilt(root):
        pass
```

Example Usage:

```
Example Input Tree #1:


     1
    / \
   2   3

Input: root = 1
Expected Output: 1
Explanation
Tilt of node 2: |0 - 0| = 0 (no children)
```

```
Example Input Tree #2:

      4
     / \
    2   9
   / \   \
  3   5   7

Example Input: root = 4
Expected Output: 15
Tilt of node 3 : |0-0| = 0 (no children)
Tilt of node 5 : |0-0| = 0 (no children)
Tilt of node 7 : |0-0| = 0 (no children)
Tilt of node 2 : |3-5| = 2 (left subtree is just left child, so sum is 3; right subtree is ju
Tilt of node 9 : |0-7| = 7 (no left child, so sum is 0; right subtree is just right child, so
Tilt of node 4 : |(3+5+2)-(9+7)| = |10-16| = 6 (left subtree values are 3, 5, and 2, which su
Sum of every tilt : 0 + 0 + 0 + 2 + 7 + 6 = 15
```

▶ ✨ **AI Hint: Absolute Value**

▶ 💡 **Hint: Choosing your Traversal Method**

Close Section

▸ Problem Set Version 2

▸ Problem Set Version 3