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

Session 1: Binary Trees

Session Overview

Students are introduced to foundational and complex tasks involving binary trees and binary search trees (BSTs). They will engage in constructing trees, manipulating tree structures, and understanding tree properties through a variety of exercises. The problems encourage exploring key operations such as inserting and removing nodes, checking tree balance, finding specific nodes, and traversing trees in various orders. This session aims to deepen students' understanding of tree algorithms, enhancing their ability to analyze and implement data structures efficiently.

> 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.

Your 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 1

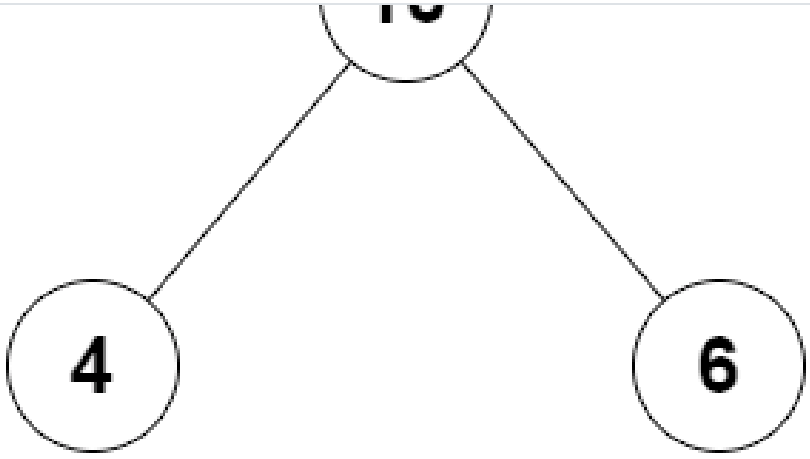
💡 Unit 8 Cheatsheet

To help your learning journey with binary trees, we've put together a guide to common concepts and syntax you will use throughout Unit 8 breakout problems. Use this cheatsheet as a quick reference guide as you work through the problems below.

▼ Problem Set Version 1

Problem 1: Build a Binary Tree I

Given the following `TreeNode` class, create the binary tree depicted in the image below.




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

▶ ✨ AI Hint: Binary Trees

Problem 2: 3-Node Sum I

Given the `root` of a binary tree that has exactly `3` nodes: the root, its left child, and its right child, return `True` if the value of the root is equal to the sum of the values of its two children. Return `False` otherwise.

Evaluate the time complexity of your function.

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

def check_tree(root):
        pass
```

Example Usage:

```
Example Input Tree #1:
  10
 /  \
4    6
Input: root = 10
Expected Output: True
```

```
 / \
3   1
Input: root = 5
Expected Output: False
```

Problem 3: 3-Node Sum II

Given the `root` of a binary tree that has at most `3` nodes: the root, its left child, and its right child, return `True` if the value of the root is equal to the sum of the values of its two children. Return `False` otherwise.

Evaluate the time complexity of your function.

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

def check_tree(root):
        pass
```

Example Usage:

```
Example Input Tree #1:
  10
 /
10
Input: root = 10
Expected Output: True

Example Input Tree #2:
  5
 / \
3   2
Input: root = 5
Expected Output: True

Example Input Tree #3:
  5
   \
    2
Input: root = 5
Expected Output: False

Example Input Tree #4:
Empty Tree (None)
Input: root = None
```

Problem 4: Find Leftmost Node I

Given the `root` of a binary tree, write a function that finds the value of the left most node in the tree.

Evaluate the time complexity of your function.

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

def left_most(root):
        pass
```

Example Usage:

```
Example Input Tree #1:

      1
     / \
    /   \
   2     5
  / \
 4   3

Input: root = 1
Expected Output: 4

Example Input Tree #2:

     1
      \
       2
      /
     3

Input: root = 1
Expected Output: 1

Example Input Tree #3:

Input: root = None
Output: None
```

If you implemented the previous `left_most()` function iteratively, implement it recursively. If you implemented it recursively, implement it iteratively.

Evaluate the time complexity of the function.

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

def left_most(root):
        pass
```

Example Usage:

```
Example Input Tree #1:

      1
     / \
    /   \
   2     5
  / \
 4   3

Input: root = 1
Expected Output: 4

Example Input Tree #2:

     1
      \
       2
      /
     3

Input: root = 1
Expected Output: 1

Example Input Tree #3:

Input: root = None
Output: None
```

Problem 6: In-order Traversal

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

def inorder_traversal(root):
        pass
```

Example Usage:

```
Example Input Tree #1:
     1
      \
       2
      /
     3

Input: root = 1
Expected Output: [1,3,2]

Example Input Tree #2 :

Input: root = None
Output: []

Example Input Tree #3:
    1

Input: root = 1
Output: [1]
```

▶ ✨ AI Hint: Traversing Trees

Problem 7: Binary Tree Size

Given the `root` of a binary tree, write a function `size()` that returns the number of nodes in the binary tree.

Evaluate the time complexity of your function.

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

```
        pass
```

Example Usage:

```
Example Input Tree #1:

      4
     / \
    /   \
   2     5
  / \
 1   3

Input: root = 4
Expected Output: 5

Example Input Tree #2:

Empty tree (None)
Input: root = None
Expected Output: 0
```

Problem 8: Binary Tree Find

Given a `value` and the `root` of a tree, write a function `find()` that returns `True` if there is a node with the given `value` in the tree. Assume the tree is balanced.

Evaluate the time complexity of your solution.

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

def find(root, value):
        pass
```

```
Example Input Tree #1:

      1
     / \
    /   \
   2     5
  / \
```

```
Expected Output: True

Example Input Tree #2:

      1
     / \
    /   \
   2     5
  / \
 4   3

Input: root = 1, value = 10
Expected Output: False
```

▶ ✨ AI Hint: Balanced Trees

Problem 9: Binary Search Tree Find

Given a `value` and the `root` of a binary search tree, write a function `find_bst()` that returns `True` if there is a node with the given `value` in the tree. Assume the tree is balanced.

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

def find_bst(root, value):
        pass
```

```
# Example Input Tree #1:

      4
     / \
    /   \
   2     5
  / \
 1   3

Input: root = 4, value = 5
Expected Output: True

Example Input Tree #2:
```

```
    /   \
   2     5
  / \
 1   3

Input: root = 4, value = 10
Expected Output: False
```

▶ ✨ AI Hint: Binary Search Trees

Problem 10: BST Descending Leaves

Given the `root` of a binary search tree, write a function `descending_leaves()` that returns a list of the values of all leaves in the BST in descending order. Assume the tree is balanced.

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

def descending_leaves(root):
        pass
```

```
Example Input Tree #1:

      4
     / \
    /   \
   2     5
  / \
 1   3

Input: root = 4
Expected Output: [5, 3, 1]

Example Input Tree #2:
 10

Input: root = 4
Expected Output: [10]
```

- **Problem Set Version 2**
- **Problem Set Version 3**