TREE EXERCISES

In this series of exercises, we'll get you up and running with arrays and strings.

In the guide, we will quickly highlight the key concepts that you need to know for your interview. If you're unfamiliar with any of these, I highly recommend taking some time to review them.

Then, I've provided you with a series of exercises to help you get up and running quickly. I recommend setting aside 2-3 days to work through these exercises. Simply start from the top and work through them as much as you have time.

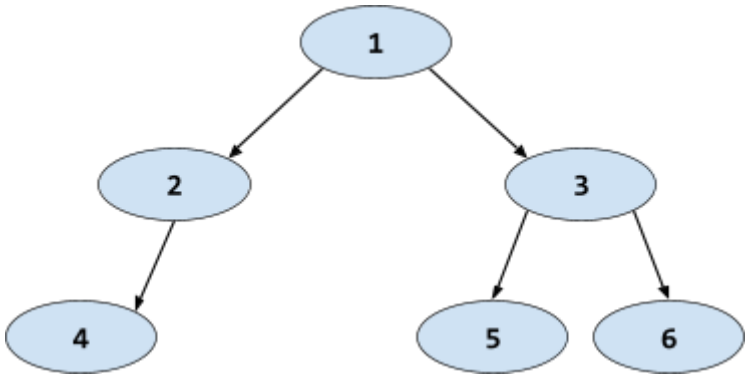
Make sure to track the time you spend here in your Interview Prep Tracker. You can categorize this as "Study" time.

For these exercises, we have complete code samples and templates available here.

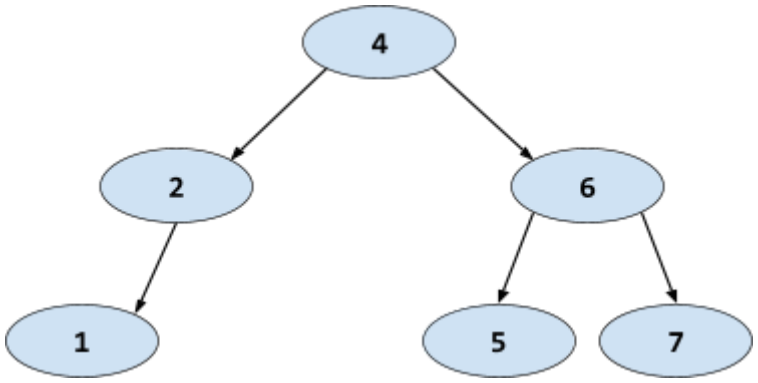
TABLE OF CONTENTS

KEY TREE TERMINOLOGY

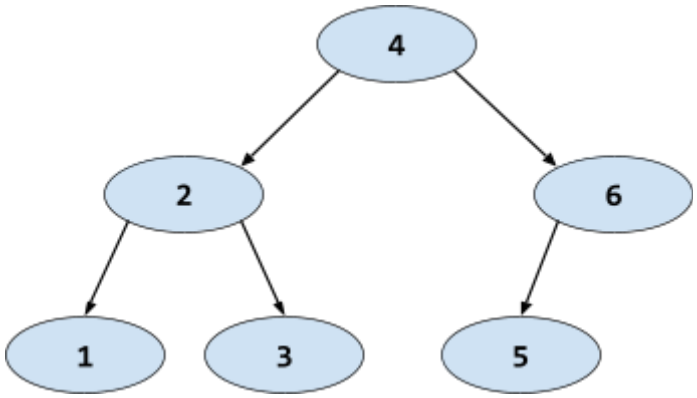
- **Binary Tree**
  A binary tree is a simple tree structure where each node of the tree has at most 2 children. There are no restrictions on what data can be in the tree or the ordering of that data.



- **Binary Search Tree (BST)**
  A binary tree where nodes in the left subtree of a node only contain values less than or equal to the root node and nodes in the right subtree of a node only contain values greater than the root node.

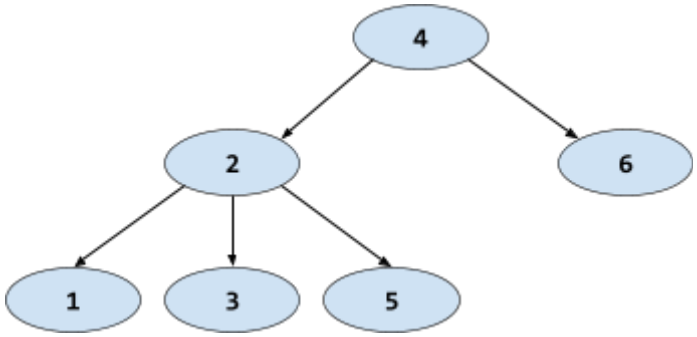


- **Complete Binary Tree**
  A binary tree where all levels except the bottom level are completely filled and the bottom level is filled in order from left to right. The examples above are NOT complete binary trees.

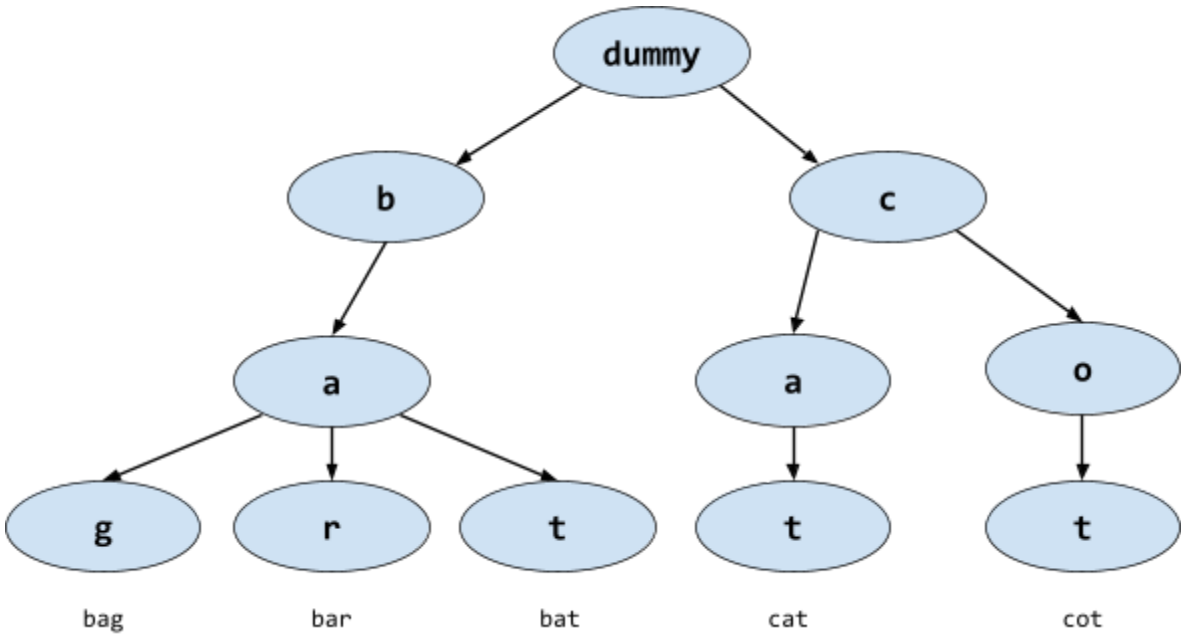


- **Balanced Binary Tree**
  A binary tree where for every node, the heights of the left and right subtrees differ by no more than 1.

- **N-ary Tree**
  A tree where each node has at most N children (N can be specified or arbitrary, ie binary tree, ternary tree, quad tree, etc).



- **Trie**
  A trie (or prefix tree) is an n-ary tree used to store string data.



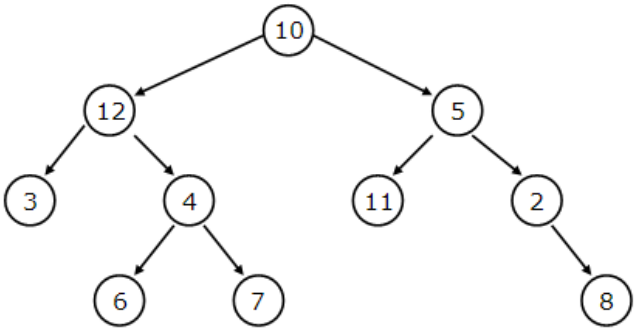
- **Tree Traversal**
  Any algorithm used to visit the nodes of a tree

KEY TREE PATTERNS

- **Accessing and Modifying Values**
  With trees, we have multiple different variations of what a tree might look like. Therefore it's important to understand how accessing and modifying values works for each type of tree.

  Rather than memorizing each type, I highly recommend looking for the similarities and differences between these patterns for each. If you understand clearly how and why they're the same/different, that is way easier than trying to memorize them as separate concepts.

- **Tree Traversals**
  There are 4 main tree traversals that we need to know how to do. It is not very important to remember the names of the different traversals, but if you know what the order is of the traversal, you should be able to implemement it.



Levelorder tree traversal
10, 12, 5, 3, 4, 11, 2, 6, 7, 8

Inorder tree traversal
3, 12, 6, 4, 7, 10, 11, 5, 2, 8

Preorder tree traversal
10, 12, 3, 4, 6, 7, 5, 11, 2, 8

Postorder tree traversal
3, 6, 7, 4, 12, 11, 8, 2, 5, 10

http://cs-people.bu.edu/tvashwin/cs112_spring09/lab06.html

- **Searching**
  Often we want to find things in a tree. If we have a BST, this becomes very easy (after all, the tree is designed for this). However, we also need to be able to search if the tree is not sorted. This basically involves doing a tree traversal.

- **Divide and Conquer**
  Trees are a data structure that is basically designed to be approached using Divide and Conquer. This simply means that we do our computation on the left subtree and right subtree separately and then combine our results together.

  For example, if you want to find the max height of the tree, you can compute the max heights of the left and right subtrees and then add 1 for the current node.

TREE EXERCISES

Exercise Set #1 - Accessing and Modifying Values

1. Implement a Binary Search Tree class with the following methods:
    a. Constructor
    b. `insert(int n)`: Insert `n` into the tree
    c. `delete(int n)`: Remove `n` from the tree
    d. `contains(int n)`: Return `true` if the tree contains the given value
    e. `serialize()`: Convert tree to a serialized form (Full Problem Definition)
    f. `deserialize()`: Convert serialized tree into a tree object (Full Problem Definition)
    g. `toString()`: You can represent this string however you like

2. Implement a Trie class with the following methods:
    a. Constructor
    b. `insert(String s)`: Insert `s` into the trie
    c. `contains(String s)`: Return true if the tree contains the given value
    d. `prefix(String pre)`: Find all strings in the trie that start with pre
    e. `toString()`: This string should just list all the strings in the trie

3. For each of your functions, compute the time and space complexity

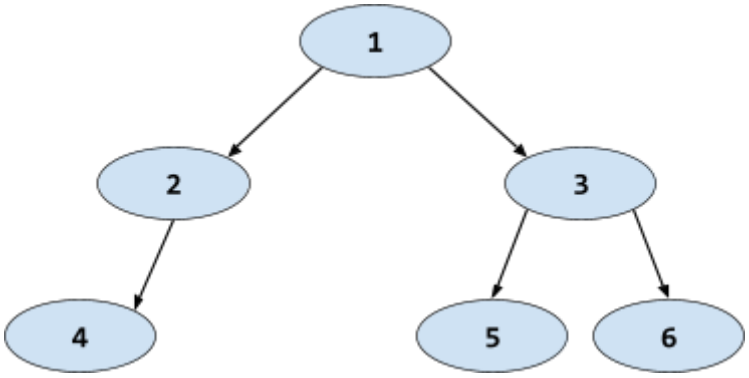
Exercise Set #2 - Tree Traversals

1. Implement each tree traversal. You should implement each traversal both recursively and iteratively:
    a. Inorder Traversal (Full Problem Definition)
    b. Preorder Traversal (Full Problem Definition)
    c. Postorder Traversal (Full Problem Definition)
    d. Level order Traversal (Full Problem Definition)

2. For each traversal, compute the time and space complexity

Exercise Set #3 - Searching

1. Implement a function that determines whether a Binary Tree contains a value.
   eg.

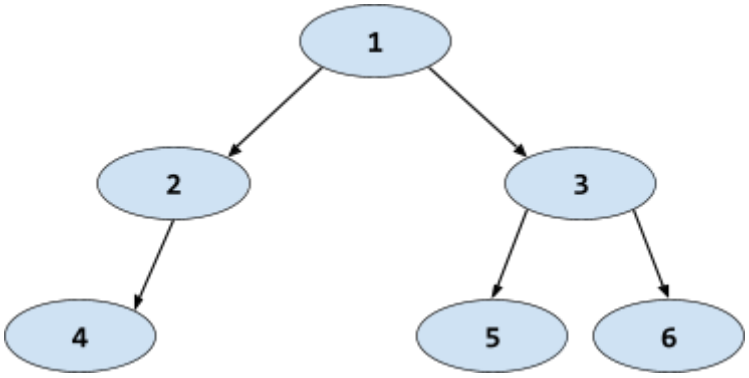


tree =

```
contains(tree, 4) = true
```

What is the time and space complexity of your solution?

3. Implement a function that determines whether a Binary Tree contains a value. If the tree contains the value, return the path to that node
   eg.



tree =

```
contains(tree, 4) = [1, 2, 4]
```

What is the time and space complexity of your solution?

Exercise Set #4 - Divide and Conquer

Solve each of the following and compute the time and space complexity.

1. Max Binary Tree Depth (Full Problem Definition)
2. Lowest Common Ancestor (Full Problem Definition)
3. Balanced Binary Tree (Full Problem Definition)
4. Merge Binary Trees (Full Problem Definition)
5. Invert Binary Tree (Full Problem Definition)
6. Diameter of a Binary Tree (Full Problem Definition)
7. Tree to Linked List (Full Problem Definition)