Worksheet
Binary Trees

Summary

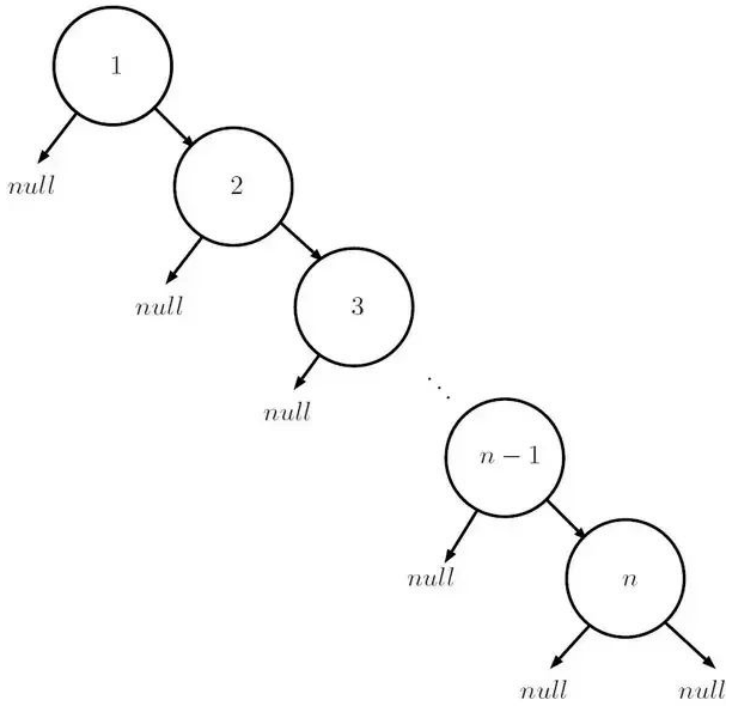
1. Subject: Data Structures
2. Topic: Binary Tree Properties, Binary Search Trees, Tree Traversals
3. Level: Easy

Binary Tree Implementation

```
class Node:

     def __init__(self, value):

          self.value = value

          self.left = None

          self.right = None
```

As you know from your data structures course, a binary tree has nodes. The topmost node is called the **root**. Every node may have at most 2 children, a left and right child.

Time Complexities



To drive some intuition behind these time complexities, let's say we have a binary tree such as the one above. Although it is linear (like an array or linked list), it still fits the definition of being a binary tree. The tree above represents our worst case scenario. *h* is the height of the tree.

- Insertion $O(h)$
- Search $O(h)$
- Deletion $O(h)$

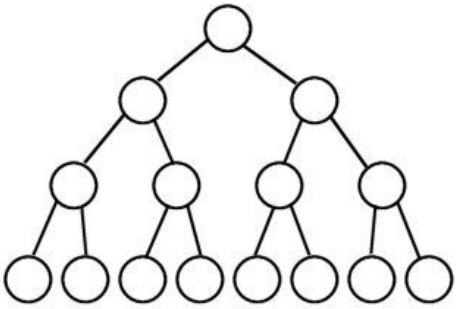
The reasoning behind these complexities is all the same. Let's say we have *n* nodes and the tree has a height *h*. If we have a tree like in the picture above, *n = h*. If we want to reach node *n* we would have to start from the root and traverse through the entire tree to reach the end so that we can insert/search/delete.

Binary Tree State

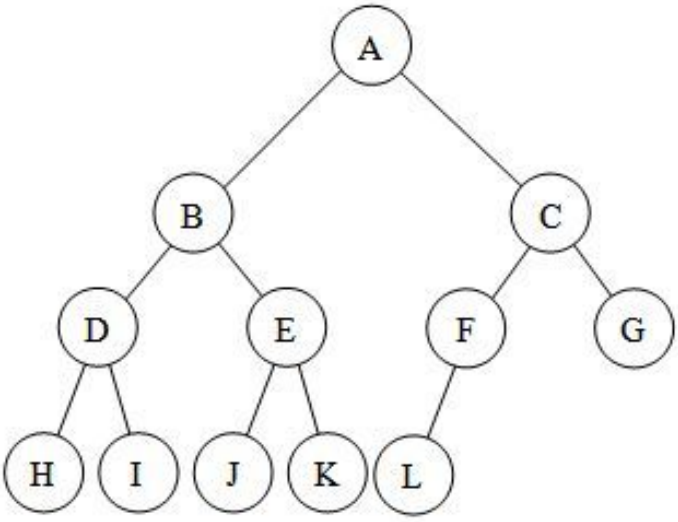
We have special names for certain states of a binary tree. These do not really show up in interview problems at all, however they are fundamental computer science definitions and it would be an embarrassment to not know them.

Full - We say we have a “full binary tree” if every node has 0 or 2 children.

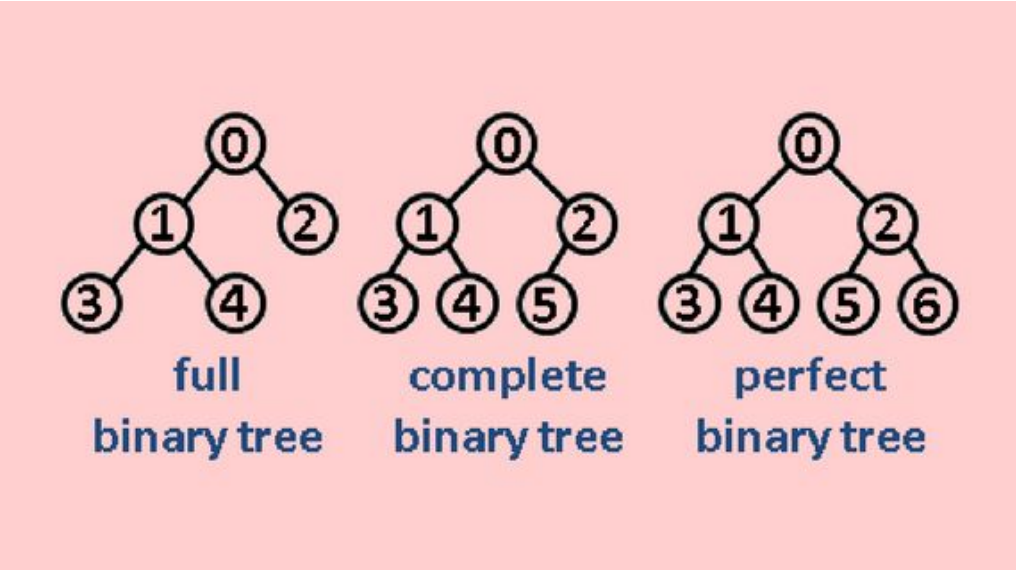
Full Binary Tree



Complete - A binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

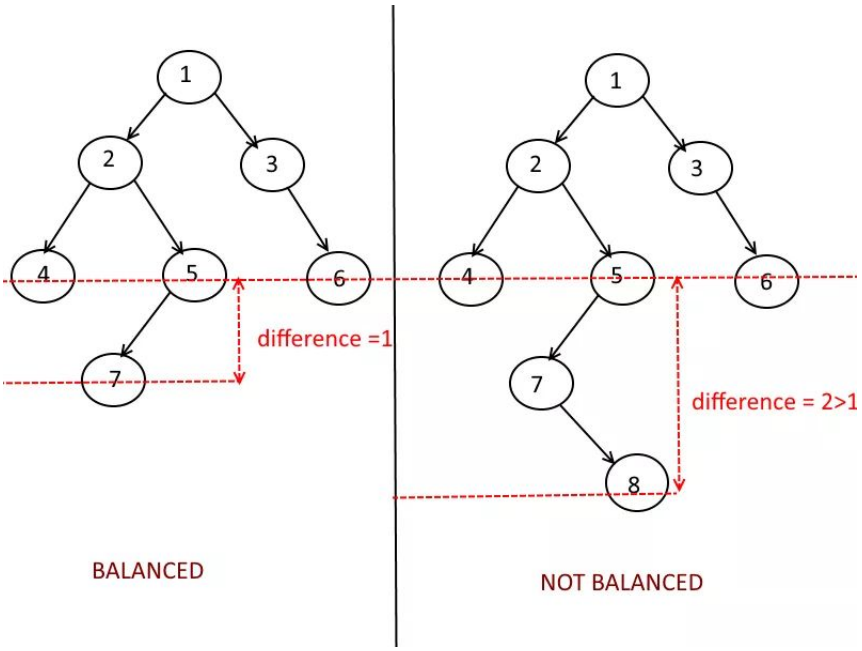


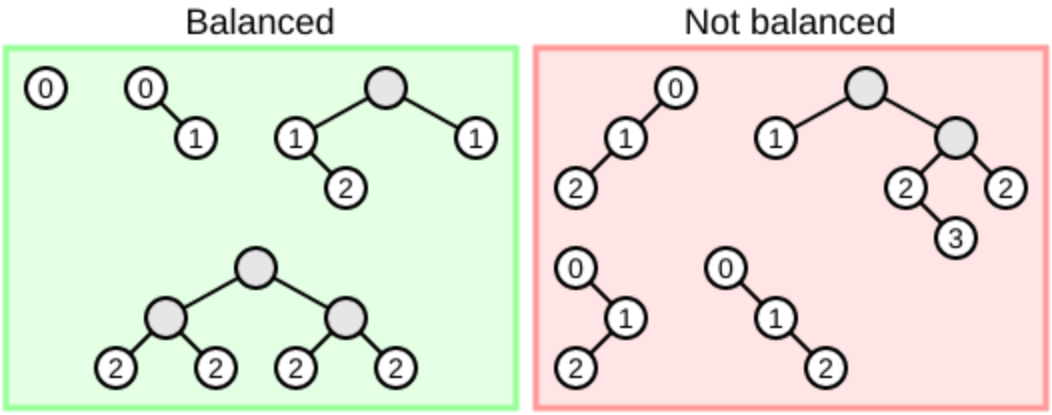
Perfect - A binary tree that is full and complete :)



Balanced Binary Trees

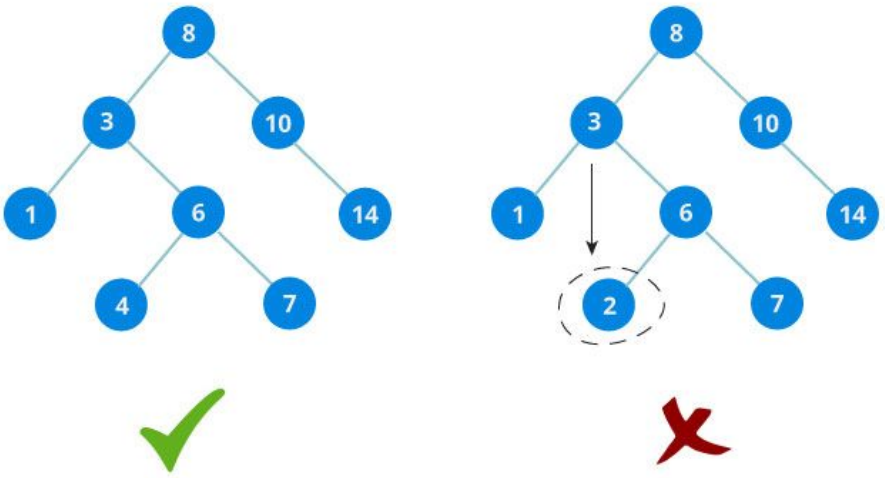
Being balanced is a very special state in a binary tree because it yields us insertion, search, and deletion in $O(log\ n)$ time as opposed to $O(h)$ where h is the height of the tree. In order for a tree to be balanced, the difference in height between any two subtrees must not differ by more than 1.





Binary Search Tree

As you know from your data structures course, a binary search tree is a binary tree where all the left children in the left subtree have smaller values than its parent, and all the right children in the right subtree have higher values than its parent.



Tree Traversals

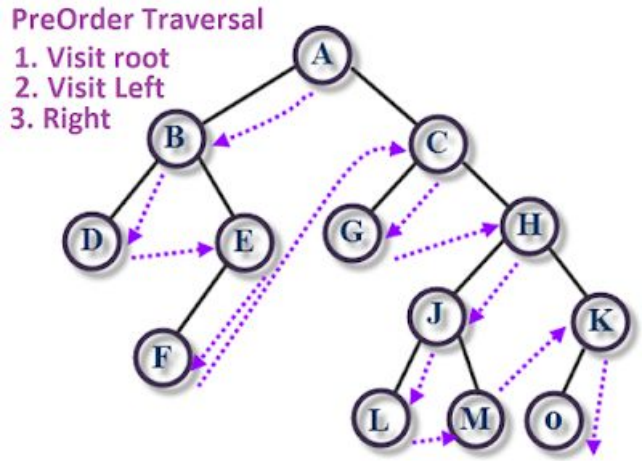
Since trees are a non-linear data structure, it is not obvious to see a way to traverse it. There are 4 standard ways to traverse trees.

1. Breadth First Search - Traverse the tree in level order
2. Depth First Search - Comes in 3 flavors
    a. Preorder Traversal
    b. Inorder Traversal
    c. Postorder Traversal

A good way to remember the order in which we traverse nodes in the 3 major traversals is to think of the positioning of when we visit the root.

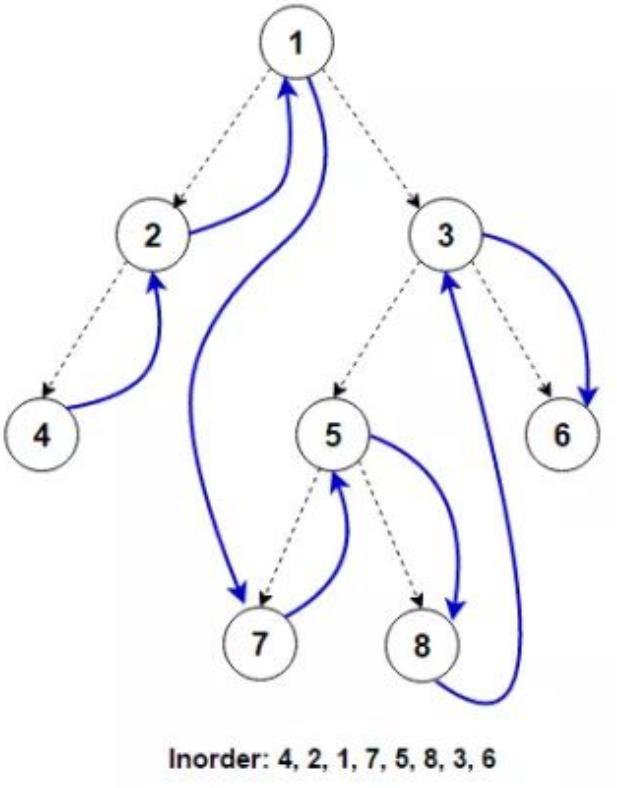
Preorder Traversal (nlr)

We traverse the tree by visiting the nodes in the order of: root node, left, right. We can abbreviate this order as (nlr) node, left, right.



Inorder Traversal (Inr)

We traverse the tree by visiting the nodes in the order of: left, root node, right.



Postorder Traversal (lrn)

We traverse the tree by visiting the nodes in the order of: left, right, root node.