# CS 008
# Lecture notes
# 4/9/24

## 1 Outline

- Trees part 2
- Binary trees

## 2 Trees Review

**Trees** are a series of nodes where one node is the root node to the tree. Every other node apart from the root node will have a parent node. All nodes are connected in some way and the connections between the nodes can be in any direction. Also, nodes usually don't know the other child nodes of the parent.

**Definitions:**

- **Parent**: A node that points to another node.
- **Child**: A node that is pointed to by a parent node.
- **Leaf**: These are nodes that do not point to another node.
- **Height & Depth**: These are the height and depth of the tree which represent the amount of nodes followed to reach the bottom for the height or a certain node for the depth. The root node is not included in the height and depth of a tree.
- **Subtree**: A portion of the tree can be also be a tree (ignoring the parent nodes of the root of the subtree to create the subtree).

## 3 Binary Tree

A binary tree is just a type of tree with up to two children.

**Full binary tree.**
Every leaf is at the same depth and every non-leaf has two children.

**Complete binary tree.**
All non-leaves (except maybe one) have two children. All leaves have the same depth $\pm 1$ and any level of the tree that is not full has nodes arranged in the left most possible spots.

### 3.1 Representation of a binary tree

We can represent a tree in memory as either an array or a linked list of nodes. A node-based binary tree should have the following node structure:

**Figure 1:**

struct node ...

### 3.2 Traversals

There are three algorithms for traversing a tree:

- **Pre-order:** (Parent, Left sub-tree, Right sub-tree)
- **In-order:** (Left sub-tree, Parent, Right sub-tree)
- **Post-order:** (Left sub-tree, Right sub-tree, Parent)

**Question:**

What is the time complexity of the search(x) operation on a random tree?

**Answer**: $O(n)$ would be the time complexity since we would have to (in the worst case) check every node for the value.

### 3.3 Binary Search Tree

Recall the **Binary Search** for a sorted array. With a sorted array, you can use binary search where you start in the middle of the array and divide the array each time by half until you find your desired result.

**Question:**

What is the time complexity of **Binary Search** on a sorted array?

**Answer**: It would have a time complexity of $O(log(n))$.

Note that a binary search only works in an ordered array (keep this in mind). For a binary search tree, the left node in a parent node would be smaller than the parent node but the right node would be the largest. So, in other words, instead of checking all nodes in the tree, we could simply look at values of the node we are currently in and move through the tree based off of the value we are searching for. This significantly shortens the time complexity of searching for a value and excludes parts of the tree we know will not contain the value.