BST Review:

- Neat for searching
- Review
    - **Traversal/Searching**
        - Basic Rule**:
            - Left , Lesser than parent ,
            - RIght, greater than parent
        - Finding 6 , (review easy look at slide)
        - Finding 11, we start at 7 (root) go all the way right (greater comparison) , and we find out there's nothing >10
    - **Insertion**
        - Case1: what if we want to insert 11,
            - We will end our search at 10, then insert 11 as the right child of 10; we do this to preserve the property "left is less" , we don't do it like heapify (easier to put things in the right place rather than to worry if its still a BST, a heap is just ok if its the root, NOW we have the condition of the whole tree, it'll be inefficient to check the whole tree if we just insert at the bottom, and not traverse then insert and make know its already correct)
        - Case2: what happens if we want to insert 1? (1 alr exists) ; can we have duplicate values in our BST?
            - **Dealing w/ duplicates (if allowed): (for this class yes!)**
                1. Arbitrarily put it to the right or left **(right for stability , during in-order traversals)** ;
                    a. **stability**; two elements of same value places don't get switched, the first one that was there always comes first
                    b. **In order traversals**; look at left subtree, the node itself, then right subtree **(LNR)**
                        i. Pre Order: LRN
                        ii. Postorder: NLR
                2. Augment nodes with *counts*: if we added another 1, ex: we can say we see 2 (1's), neater but takes up more space
                - If some don't do duplicates, replace the number w the same number
            -

- **Deletion**
    - Case 1: What if we want to delete 11?
        - Traverse to 11, then delete, same logic for traverse and insert
            - Just like heaps, deleting leaf nodes is easy
    - Case 2*: what happens if we want to delete 2?
        - Traverse to 2
            - We want to find the ***smallest value greater than 2*** (right then left), and put it over there which minimizes the work to see what's in place (preserves BST property)
                - Swap 3 and 2 , and delta leaf as usual, (no consequences and not have to worry abt BST property)
    - Case 3*: Now what if we want to delete 3 after?
        - 4 does not have any children, any 3 only has 1 child, we can link 3's parents and child

BST Time Complexity*

In this class if theta is defined we should use it
Data structure aht literally just helps w binary search

Searching	Inserting	Deleting
Best: Θ(1) (returned first)	Best: Θ(1)* *Best case is very rare for inserting a deleting (the first element), after that we have to traverse	Best: Θ(1) Very rare!
Avg: Θ(logn)	**Avg: Θ(logn)** Similar to searching, but just inserting which is a constant amt of time and doesn't change answer ; we have a decision at **every level,** which **eliminates half the possibilities** (mathematically translates to log)	**Avg: Θ(logn)**

Worst: Θ(n) (returned last)	Worst: Θ(n)	Worst: Θ(n) Intuition: even though it swaps, the intuition is that it will simplify down; - Scanning through to the end, will look like arr?; ex: **<u>skewed BST</u>**

Regular Trees

Min max heaps, bst have special properties, if it doesn't have a property, it's just a tree

- Nodes can have more than 2 sub children

3/6 Lab 4 & 5
Lab 4 & 5 important for knowing trees (google interview also)