AVL Tree

- Problems with BST: The order in which elements are inserted affects the height of the tree. If elements are inserted in either ascending or descending order, then the tree is completely imbalanced. The more imbalanced a tree is, the more comparisons we have to make. Therefore, our goal is to create a BST with minimum height.
    - This can be achieved by a complete BST
        - A complete BST is a BST where every level is full except possibly the last one.
    - This ensures at most log(n) comparisons.
    - However, it is cumbersome to build such a tree from a list of numbers
    - Here come AVL Trees!
- An AVL tree is an approximately balanced BST tree
    - Maintains a balance factor in each node
    - The balance property of a given node states that the maximum difference between the height of the left subtree and the height of the right subtree is 1.
    - Each time a node is inserted, the balance property is checked. If it is not satisfied, the tree must be rotated.
    - There are four cases that warrant rotation
        - Case 1: Left-Left Insertion
            - If Z is the node of imbalance, left-left rotation is caused when the inserted node falls in the left subtree of the left child of Z
            - Identify the unbalanced node (let's call it Z).
            - Let Y be the left child of Z, and X be the left child of Y.
            - Perform a right rotation:
                - Make Y the new root.
                - Move Z to be the right child of Y.
                - Assign Y's right child as Z's left child.
        - Case 2: Left-Right Insertion
            - Occurs when you insert the new node into the right subtree of the left child of Z
            - Identify the unbalanced node (Z).
            - Look at Z's left child (Y), which has a right child (X).
            - Step 1: Move X up and Y down
                - X takes Y's position.
                - Y becomes X's left child.
                - If X had a left subtree, it becomes Y's right subtree.
            - Step 2: Move X up and Z down

        - X takes Z's position.
        - Z becomes X's right child.
        - If X had a right subtree, it becomes Z's left subtree.
- Case 3: Right-Left Insertion
    - Occurs when you insert the new node into the left subtree of the right child of Z
    - Identify the unbalanced node (Z).
    - Look at Z's right child (Y), which has a left child (X).
    - Step 1: Move X up and Y down
        - X takes Y's position.
        - Y becomes X's right child.
        - If X had a right subtree, it becomes Y's left subtree.
    - Step 2: Move X up and Z down
        - X takes Z's position.
        - Z becomes X's left child.
        - If X had a left subtree, it becomes Z's right subtree.
- Case 4: Right-Right Insertion
    - Occurs when the new node is inserted into the right subtree of the right child of Z
    - Identify the unbalanced node (Z).
    - Let Y be the right child of Z, and X be the right child of Y.
    - Perform a left rotation:
        - Make Y the new root.
        - Move Z to be the left child of Y.
        - Assign Y's left child as Z's right child.