BST Rotation Summary

- Four kinds of rotations (L, R, LR, RL)
- All rotations are local (subtrees are not impacted)
- All rotations are constant time: O(1)
- BST property maintained

**GOAL:**

We call these trees:
AVL Trees

Strategy to Maintain Balance
- Detecting Imbalance

- Store/Update Height

- What Rotation to Use
AVL Tree Rotations

Four templates for rotations:
Theorem:
If an insertion occurred in subtrees $t_3$ or $t_4$ and a subtree was detected at $t$, then a __________ rotation about $t$ restores the balance of the tree.

We gauge this by noting the balance factor of $t$->right is ______.
Finding the Rotation on Insert

**Theorem:**
If an insertion occurred in subtrees $t_2$ or $t_3$ and a subtree was detected at $t$, then a __________ rotation about $t$ restores the balance of the tree.

We gauge this by noting the balance factor of $t$->right is _______.

![Diagram of a tree with nodes $t_1, t_2, t_3, t_4,$ and $t$.]
Insertion into an AVL Tree

```c
struct TreeNode {
    T key;
    unsigned height;
    TreeNode *left;
    TreeNode *right;
};

_insert(6.5)```
Insertion into an AVL Tree

Insert (pseudo code):
1: Insert at proper place
2: Check for imbalance
3: Rotate, if necessary
4: Update height

```c
struct TreeNode {
    T key;
    unsigned height;
    TreeNode *left;
    TreeNode *right;
};
```

_\text{insert}(6.5)_
template <typename K, typename V>
void AVL<K, D>::_insert(const K & key, const V & data, TreeNode *& cur) {
    if (cur == NULL) { cur = new TreeNode(key, data); }
    else if (key < cur->key) { _insert(key, data, cur->left); }
    else if (key > cur->key) { _insert(key, data, cur->right); }
    _ensureBalance(cur);
}
```cpp
template <typename K, typename V>
void AVL<K, D>::_ensureBalance(TreeNode *& cur) {
    // Calculate the balance factor:
    int balance = height(cur->right) - height(cur->left);

    // Check if the node is currently not in balance:
    if (balance == -2) {
        int l_balance =
            height(cur->left->right) - height(cur->left->left);
        if (l_balance == -1) { ____________________________; }
        else { ____________________________; }
    } else if (balance == 2) {
        int r_balance =
            height(cur->right->right) - height(cur->right->left);
        if (r_balance == 1) { ____________________________; }
        else { ____________________________; }
    }

    _updateHeight(cur);
};
```
AVL Tree Analysis

We know: insert, remove and find runs in: ____________.

We will argue that: $h = ____________$. 
AVL Tree Analysis

Definition of big-O:

...or, with pictures: