Final Project/Final Exam
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);
}
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 current 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: $h$ is ___________.
AVL Tree Analysis

Definition of big-O:

...or, with pictures:

\[ n, \text{number of nodes} \]

\[ h, \text{height} \]
AVL Tree Analysis

• The height of the tree, $f(n)$, will always be less than $c \times g(n)$ for all values where $n > k$. 
AVL Tree Analysis

h, height

n, number of nodes

n, number of nodes

h, height
AVL Tree Analysis

- The number of nodes in the tree, $f^{-1}(h)$, will always be greater than $c \times g^{-1}(h)$ for all values where $n > k$. 

\[ f^{-1}(h) > c \times g^{-1}(h) \]
Plan of Action

Since our goal is to find the lower bound on $n$ given $h$, we can begin by defining a function given $h$ which describes the smallest number of nodes in an AVL tree of height $h$: 
Simplify the Recurrence

\[ N(h) = 1 + N(h - 1) + N(h - 2) \]
State a Theorem

**Theorem:** An AVL tree of height $h$ has at least ________.

**Proof:**
I. Consider an AVL tree and let $h$ denote its height.
II. Case: _______________

An AVL tree of height ____ has at least ____ nodes.
Prove a Theorem

III. Case: ______________

An AVL tree of height _____ has at least _____ nodes.
Prove a Theorem

By an Inductive Hypothesis (IH):

We will show that:

An AVL tree of height _____ has at least _____ nodes.
Prove a Theorem

V. Using a proof by induction, we have shown that:

...and inverting:
Summary of Balanced BST

Red-Black Trees
- Max height: $2 \times \lg(n)$
- Constant number of rotations on insert, remove, and find

AVL Trees
- Max height: $1.44 \times \lg(n)$
- Rotations:
Summary of Balanced BST

**Pros:**

- Running Time:
  - Improvement Over:

- Great for specific applications:
Summary of Balanced BST

Cons:
- Running Time:

- In-memory Requirement:
Red-Black Trees in C++

C++ provides us a balanced BST as part of the standard library:

```cpp
std::map<K, V> map;
```
Red-Black Trees in C++

\[ V & \, \text{std::map}<K, \, V>::\text{operator[]}\left( \, \text{const} \, K & \right) \]
Red-Black Trees in C++

\[
V & \& \text{std::map}<K, V>::\text{operator[]}(\text{const } K &)
\]

\[
\text{std::map}<K, V>::\text{erase}(\text{const } K &)
\]
Red-Black Trees in C++

```cpp
iterator std::map<K, V>::lower_bound( const K & );
iterator std::map<K, V>::upper_bound( const K & );
```
## Every Data Structure So Far

<table>
<thead>
<tr>
<th></th>
<th>Unsorted Array</th>
<th>Sorted Array</th>
<th>Unsorted List</th>
<th>Sorted List</th>
<th>Binary Tree</th>
<th>BST</th>
<th>AVL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Find</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Insert</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Remove</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Traverse</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>