Insert Operations

**Question 1.** What would happen if we were to insert the value 14 into the above tree? Is it still an AVL tree?

**Question 2.** What if we insert the value 34 into it instead? If we follow the standard binary search tree insertion algorithm, is it still an AVL tree?

After inserting a node to a binary search tree, we apply the following procedure:

Start at the newly-inserted node and walk up to the root, checking if each node is balanced (the height-balance rule applies to this node). If a node is unbalanced, rotate the subtree rooted at that node. Rotate the following three nodes:

1. Let \( z \) be (a pointer to) the first unbalanced node on the way up.
2. Let \( y \) be the child of \( z \) with greater height (hint: this is always an ancestor of the node you inserted. Why?). Why are ties impossible?
3. Let \( x \) be the child of \( y \) with greater height (this is always an ancestor of the node you inserted, or the node itself. Why?). Why are ties impossible?

When \( x, y, z \) form a zig-zag pattern, we do a **double rotation**. Otherwise we do a **single rotation**. The rotations are pictured on the last page.

**Question 3.** After doing a rotation, the tree is guaranteed to be balanced, and we can stop. Why?
**Question 4.** Starting from the original tree, insert the value 34. What is the resulting AVL tree?

**Question 5.** Starting from the original tree, insert the value 54. What is the resulting AVL tree?

### Deleting from an AVL Tree

To delete from an AVL tree, follow the same procedure as removal from a binary search tree. Then, starting at the node that was removed, move up to the root, recalculating heights if necessary. For each node \( z \), if it is unbalanced, rotate the subtree rooted at that node:

1. Let \( z \) be (a pointer to) the unbalanced node we found.
2. Let \( y \) be the child of \( z \) with greater height (hint: this is never an ancestor of the node you deleted. Why?). Why are ties impossible?
3. Let \( x \) be the child of \( y \) with greater height. In the event of a tie, choose \( x \) so that we’ll have a single rotation instead of a double rotation.

Then perform the same rotation you would do for that formation on insertion. Unlike insertion, however, this only fixes the problem *locally* – it might be unbalanced higher up. You need to continue this until the tree is balanced *globally.*

**Question 6.** Starting from the original tree, what does it look like if we delete 88?

**Question 7.** Starting from the original tree, what does it look like if we remove 32?

**Question 8.** Starting with an empty AVL tree, do the following operations, in sequence:

- **INSERT:** 1, 2, 3, 12, 9, 13, 7, 4, 6, 5, 8
- **DELETE:** 4, 1
- **INSERT:** 1, 14, 11
- **INSERT:** 3, 13, 12, 11, 14, 2, 3, 7, 8, 9

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left rotate on z</strong></td>
<td></td>
</tr>
<tr>
<td><img src="before1.png" alt="Diagram" /></td>
<td><img src="after1.png" alt="Diagram" /></td>
</tr>
<tr>
<td><strong>Right rotate on z</strong></td>
<td></td>
</tr>
<tr>
<td><img src="before2.png" alt="Diagram" /></td>
<td><img src="after2.png" alt="Diagram" /></td>
</tr>
<tr>
<td><strong>Right rotate on y, then left rotate on z</strong></td>
<td></td>
</tr>
<tr>
<td><img src="before3.png" alt="Diagram" /></td>
<td><img src="after3.png" alt="Diagram" /></td>
</tr>
<tr>
<td><strong>Left rotate on y, then right rotate on z</strong></td>
<td></td>
</tr>
<tr>
<td><img src="before4.png" alt="Diagram" /></td>
<td><img src="after4.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>