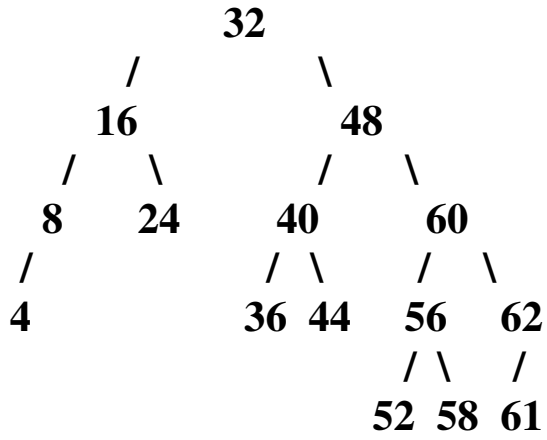
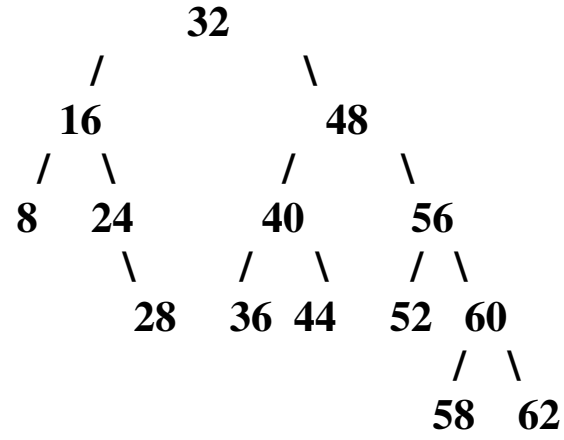
Tracing through the code, we find the first place an imbalance occurs tracing up the ancestry of the node storing 61 is at the noce storing 56. This time, we have that node A stores 56, node B stores 60, and node C stores 62. Using our restucturing algorithm, we find the tallest grandchild of 56 to be 62, and rearrange the tree as follows:



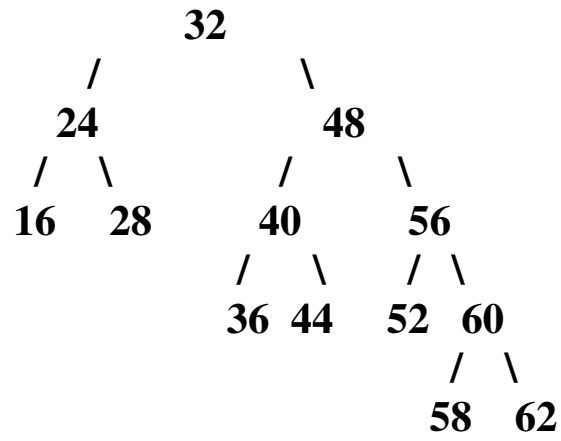
T_0 is the subtree rooted at 52, T_1 is the subtree rooted at 58, T_2 is the subtree rooted at 61, and T_3 is a null subtree.

3) ***For this example, we will delete the node storing 8 from the AVL tree below:***



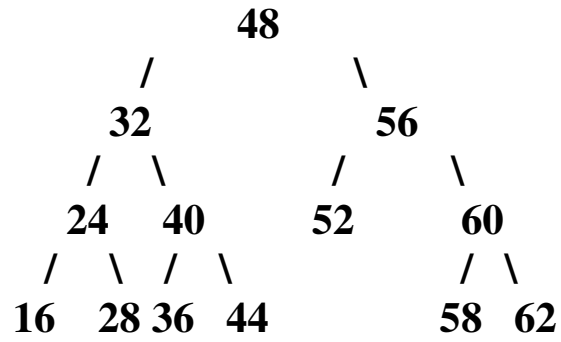
Tracing through the code, we find that we must first call the rebalance method on the parent of the deleted node, which

stores 16. This node needs rebalancing and gets restructured as follows:

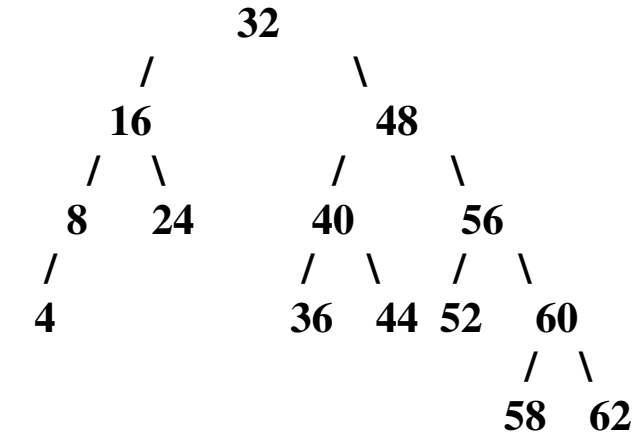


Notice that all four subtrees for this restructuring are null, and we only use the nodes A, B, and C. Next, we march up to the parent of the node storing 24, the node storing 32. Once again, this node is imbalanced. The reason for this is that the restructuring of the node with a 16 reduced the height of that subtree. By doing so, there was in INCREASE in the difference of height between the subtrees of the old parent of the node storing 16. This increase could propogate an imbalance in the AVL tree.

When we restructure at the node storing the 32, we identify the node storing the 56 as the tallest grandchild. Following the steps we've done previously, we get the final tree as follows:



4) ***The final example, we will delete the node storing 4 from the AVL tree below:***



When we call rebalance on the node storing an 8, (the parent of the deleted node), we do NOT find an imbalance at an ancestral node until we get to the root node of the tree. Here we once again identify the node storing 32 as node A, the node storing 48 as node B and the node storing 56 as node C. Accordingly, we restructure as follows: