Task 2 Discussion

1. **Explain the high-level idea of your algorithm (this should only be a few sentences)**
   Check whether self or other is the AVL tree with smaller keys by comparing the keys of the root nodes. Then, remove the corrupted items from the smaller tree and create a new temporary node. Assign the tree with smaller keys to the left child of the new node and the tree with bigger keys to the right child of the new node. After that, delete the temporary node and rebalance the tree.

2. **Give the complexity of your implementation of uncorrupted merge. Explain why it has this complexity. Be sure to define any variables you use in your complexity (other than the ones defined in section 2.6)**
   The time complexity is O(C log(N) + N) where in the number of items in the AVLTree with larger keys (either self or other) and C is the number of items in corrupted. First, deleting the corrupted keys will give a time complexity of O(C log M), where M is the number of nodes in the smaller AVL tree. This comes from the time complexities of looping through the corrupted keys, O(C), and the deletion, O(log N). Since the temporary node will be deleted, it will result in O(N + M), because the trees have been merged. Lastly the rebalancing will give O(N + M) as it goes through the merged tree.

   All in all, the time complexity is
   $$O(C\log M + N + M + N + M) = O(C\log M + 2N + 2M)$$
   $$= O(C\log M + N + M)$$
   Since M is smaller than N we can reduce it to
   $$O(C\log M + N + M) = O(C\log M + N + N)$$
   $$= O(C\log M + N)$$

3. **Justify that your algorithm works. In other words, explain why the output is a valid AVL tree.**
   The algorithm goes through the tree recursively and uses the given rebalancing function to rebalance the unbalanced sub trees. Heigh and balance is updated throughout the process.

4. **Suppose we now relax the constraint that the number of items in corrupted is much smaller than the number of items in t1 and t2. Does this change the relative efficiencies of Nathan's approach and your solution? If so, does Nathan's approach ever become more efficient, and roughly when would this occur?**
   If the items in corrupted is much smaller, my approach would become more effective than Nathan's. The amount of insertions Nathan's approach would increase and make its even less effective than before. Nathan's approach would be more effective if the number corrupted items is large which would decrease the number of insertions needed to merge both tables.