Name: Poon Yeong Shian
ID: 30696003

1. Explain the high level idea of your algorithm (this should only be a few sentences)

- Firstly, we let the AVL tree with all items smaller than other avl tree be Left_AVL and the other be Right_AVL.
- Secondly, we create a new Root_AVL avl tree with a temporary key (which is the median root key of Left_AVL and root key of Right_AVL) at its node and we assign the Left_AVL to be the left sub-avl tree and Right_AVL to be the right sub-avl tree.
- Thirdly, we remove corrupted one by one from the left sub-avl tree.
- Fourthly, we find the logical successor for the root of the Root_AVL avl tree and we delete it from the right sub-avl tree.
- Lastly, we replace root key of the Root_AVL avl tree with the successor key we found earlier.

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)

- L is the corrupted items in one of the AVL tree. N is the number of nodes in self AVL tree.
- The time complexity when merge the lesser key sub-AVL tree and more key sub-AVL tree to a big AVL tree is O(1) because we just need to assign these two sub-AVL tree to left and right of the new AVL tree we created.
- The time complexity to remove all the corrupted item from the left sub-AVL tree of height log N is O(L log N).
- The time complexity for us to find and delete the successor node for the root in this new AVL tree is O(log N).
- The time complexity for us to change the root key to the successor key is O(1).
- So, the overall complexity is O(L log N).

3. Justify that your algorithm works. In other words, explain why the output is a valid AVL tree.

- When we delete the successor node of the root from the right AVL-sub tree, we will always rebalance that subtree.
- The output is a valid AVL tree because height difference of the child nodes does not exceed 1.

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?

- When the number of corrupted items become much smaller, efficiencies of Nathan's approach will become the same as you delete less item but insert more item to the AVL tree one by one. My solution is more efficient because rather than inserting the item one by one to the tree, I just need to make t1 or t2 to become the sub-AVL tree of my large AVL tree.