The worst case performance of the insert_element function is O(log_2 n) which happens when inserting at a leaf node. This is because a balanced tree makes searching much faster as it cuts the number of possible nodes in half at each step. Had the tree not been balanced, this would be linear time since the recursive_insert_element function is inherently constant, but calls itself recursively, making it linear. However, the __balance function (which runs in constant time since it only relies on other constant functions) balances the tree which in turn makes the run time for the insert_element function logarithmic.

The remove function acts in an almost identical manner to the insert function, so it also runs in O(log_2 n) time.

The to_list method runs in O(n^2) time in the worst case. This is because it is inherently a constant time function (appending to the end of a list is a constant time method) but it is linear in the worst case when there is not enough space in the array (since you have to copy the values and transfer them to make space which takes linear time) that calls itself recursively, which makes it linear*linear=quadratic.

The sorting method for this project (inserting n elements into a AVL tree and then calling the to_list method) actually runs in O(n*log_2 n) time. The reason this is the case is in the following points:

-Inserting a single element into a AVL tree takes O(log_2 n)
 -We are inserting n elements into an AVL tree (hence, this takes O(n*log_2 n) run time.
 -Calling to_list is a quadratic time function which happens separately from the above two (so we add).

Therefore, the run time for sorting should be O(max(n*log_2 n, n^2))=O(n^2) since the first two bullets are separate from the third so we do addition.

The sorting does not depend on the type of objects being sorted.This is because the only thing that matters in sorting is the size of the tree, not the type of comparison operations being used.

I have done several things to check that all my methods are working properly. Firstly, I made sure that my code passed all of the gradescope test cases. This should give me good reason to believe that my code is functioning properly. I also tested my methods in the main section of my file to double check. I also wrote a ton of test cases in my test case file to check everything. Also, quoting the specifications, "the three traversals (in-order, post-order, and pre-order) uniquely identify a binary search tree. No two unequal trees share all three traversal orderings. Ensure that your traversals work correctly and use the combination of all three of them to test the structure of the tree after insertion and removal operations." I pretty much used this to the best of my ability to test all possibilities. For every test case I wrote, I tested the str, height, and three traversal functions.

I tested inserting and removing from trees of various heights (0-3) and I also tested the balancing function that they both call by inserting and removing elements that force a tree to become unbalanced. I checked that all these methods changed the tree in the expected manner by making sure that all three traversals were correct.

I also tried my best in the test cases to check for corner cases such as inserting copies of existing elements into a tree, removing non-existent elements from a tree, removing elements from an empty tree, etc...

For the to_list method, I just inserted a bunch of random values and checked to see that they were sorted. This method follows the exact same logic as my in_order so I do not need to check the functionality a lot - I just needed to check that it returned a list instead of a string (which it does).