When comparing AVL trees and Hash tables, a number of factors can be analyzed, but when the data comes in only one thing truly matters: efficiency. AVL trees and Hash tables are both valid options for data storage and retrieval systems, but each have their own draw backs and bonuses. The BigO notation for AVL trees compared to Hash tables lends credence to the Hash tables more efficient BigO value. The O(log n) time on AVL trees is a constant and unchanging factor across all implementations of the AVL tree structure, however the Hash table's potential of a constant time look up O(1) is enticing from an efficiency standpoint.

To answer the question of which is the superior data structure, assuming an arbitrarily large data set, the Hash table beats its competitor out. By increasing the number of buckets in the table, utilizing a good hashing function, and correctly handling collisions as they occur, the O(1) look up time quickly outstrips the AVL tree. With each iteration, the AVL tree eliminates half the remaining data while the Hash table has the potential to eliminate (N-1 / N) data elements, where N is the number of buckets. This would equate to 99.999% of the data with 100,000 buckets, a similar magnitude to the number of documents in the corpus of this project.

Some comparison reinforces this fact. Consider set of 32 numbers (2^5) inserted into an AVL tree, and the same set inserted into a Hash table with 8 buckets containing AVL trees, to reach the left most node of the AVL tree or the least value stored, is a 5 operation procedure in the AVL tree, versus a 3 operation procedure for the Hash table (1 to access the table, and 2 more to progress down the AVL tree stored inside that bucket). Increase the buckets of the Hash table now, with 16 buckets it's now a 2 operation. Increasing the buckets in the Hash table to be the size of our data set, that is 32 buckets, we find a constant time of 1 operation.

If we had been able to fully implement both the AVL tree and Hash table data structures along with maintenance mode, we would have been able to supply timing data to verify the above theoretical explanation. We would have also been able to ascertain at what size, if any, the overhead of using a Hash table implementation would become less important than the speedup gained from using a Hash table.