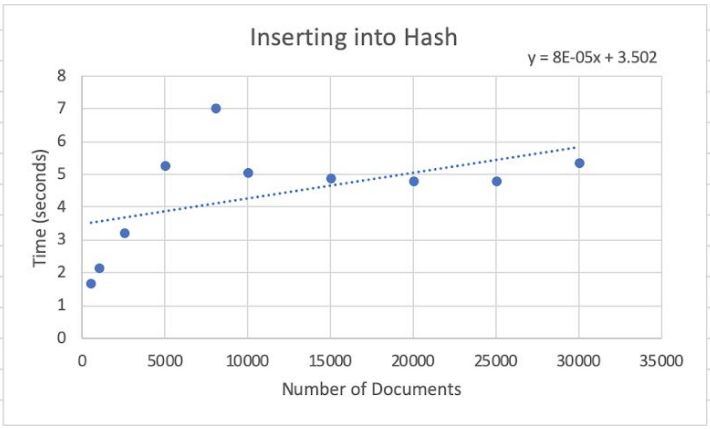
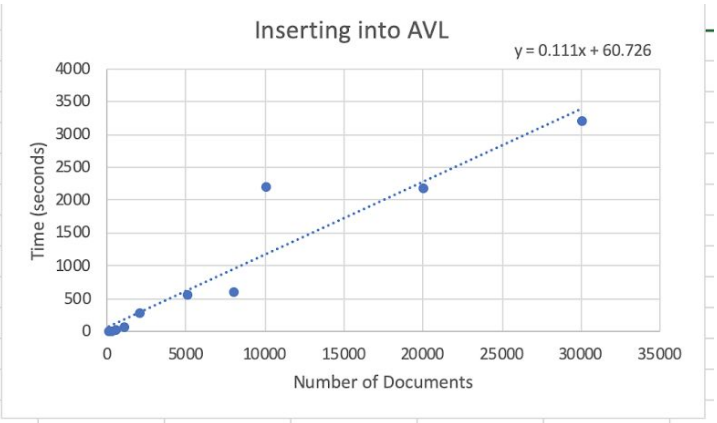
Fidelia Nawar & Annalise Sumpon
CSE2341
12/09/2019

Final Project Report

Based upon testing our program with different data sets, we came to the conclusion that the hash table has faster functionality than the AVL tree. We came to this conclusion by timing differences in parsing and indexing of small, medium, and large data sets using the two different data structures. As depicted in the graph, the speed of searching and inserting into the hash was faster than that of the AVL tree. Looking at the charts, we can see that the hash table inserts at a rate much faster than that of the AVL tree, especially for large data sets. The hash table is also consistently inserting within 4-5.5 seconds despite increase the document size by 6 times. On the other hand, AVL trees can similarly insert fast, but are not as efficient for larger data sets compared to hash tables.





Hashtable - Insert			AVL - Insert	
Number of Documents	Seconds		Number of Documents	Seconds
100	0.78		100	7
500	1.67		500	24
1000	2.14		1000	53
2500	3.21		2500	278
5000	5.27		5000	564
8000	7.03		8000	604
10000	5.05		10000	2201
15000	4.87		15000	2349
20000	4.78		20000	2189
25000	4.8		25000	3018
30000	5.35		30000	3216

Hash Table data

The efficiency for operations of hash tables are constant time close to O(1) for the operations of inserting and searching. This was beneficial for our program because by being able to quickly insert into the hash without needing to be concerned about rebalancing factors, there was a much greater time efficiency. Furthermore, having the ability to search faster was

Fidelia Nawar & Annalise Sumpon
CSE2341
12/09/2019
further beneficial for our program because of our use of an index handler to use words to search for documents.

AVL Tree data

AVL trees, on the other hand, have a logarithmic time complexity where searching, inserting, and removing all take O(log n) in average and worst cases. The operations for this data structure are more complex than the operations of the hash table simple due to the need to continuously rebalance the tree with added data. The benefit of using the AVL tree was that we were able to have persistency when inserting/removing an element from the tree in O(log N) space-and-time complexity. This allowed us not only to obtaining a new tree, but also get to retain the old tree.

Comparison

For our program, our dominant operations were *insert* and *find,* which is why using hash table provided the larger benefit. The memory footprint for hash tables are low compared to that of AVL trees because since we are performing hashing without collisions, we were not constantly increasing the size of our hash table but rather adding to the vector of values for an already existent key. On the other hand, AVL trees perform better for operations such as traversals or searching for min/max elements which wasn't necessary for the functionality of our program. Regardless of input size, we noticed that using a hash table had the faster, more consistent time complexity of inserting and searching. For the AVL tree, inserting and sorting for small and medium data sets were similarly quick and completing within under 1 and 15 seconds respectively, but for the large data set, it was taking about 50 minutes to index into the tree. For the hash table, the inserting process was occurring quickly under 7 seconds despite the significant increase in data set size.