Grant Mercer

Assignment 7

Part B

**Trie** vs **avlTree**:

In this specific program a trieTree performs much better than an avlTree. The trie’s ability to share common letters in words allows for fast lookup and a very efficient lookup and search. The avlTree on the other hand has a much lighter load on memory than the TrieTree, as each node only needs two children compared to the 26 a trie node has. This leads to a lot of unused memory in the trie structure essentially bloating it.

The trie remains a very fast lookup, insert and isPrefix having a complexity of O(M) where M is the length of the key to be inserted or searched. This means as the size of the structure is not dependent on the lookup but rather the key size, making a trie very efficient with large amounts of data. An avlTree still has a fast lookup and insert time of O(log n), but the log n will still grow to a large time as the data becomes larger and larger.

Space required for the small dictionary For each data structure:

avlTree Implementation: 1,285,872 bytes

trieTree implementation: 33,117,768 bytes

Hardware used and timing results:

For this program I used the STE||AR Research groups Hermione Cluster to measure the difference in each program. I specifically used the [Marvin node](http://stellar.cct.lsu.edu/resources/hermione-cluster/) which is 16 cores and 48GB of RAM. The timing results were:

Execution time:

avlTree:

real 0m00.654s

user 0m00.652

trieTree:

real 0m00.352s

user 0m00.304s