

The data structures that could be used to address the requirements include a binary tree, a hash table, and a simple vector. The memory requirements for the vector and binary tree are fairly straight forward using one data structure per line. The hash table, however, requires extra allocated memory to reasonably reduce the number of collisions while allowing growth. The typical run time for the hash table is O(1) which is pretty fast. The typical run time for the binary tree is O(log2(n)) which is faster than the vector with a run time of O(n). The hash table is typically the fastest.

The worse-case running time for loading the data using the vector and hash table is O(n). The worse-case running time for loading the data using the binary tree is O(log2(n) \* n). The binary tree is slower because it must reorder the data while loading.

Since the data in the binary tree is already in order, it is possible to print out the data in alphanumeric order without taking extra time. Another advantage of the binary tree is that the search run time is fairly stable whereas the vector could take from O(1) to O(n). The hash table also relies on probable distribution for an O(1) run-time with an O(n) worse-case scenario.

In terms of the Big O, printing the data in alphanumeric order using the binary tree is faster than the other algorithms. The other algorithms would require the data to be reordered. The hash table can be very fast for searching; however, it is still a probabilistic operation and requires extra memory. The vector, which is fairly straight forward, would always require a linear search which can be slow. Based on this analysis I plan to use the binary tree in my code.