# CS 300 Pseudocode Document

## Runtime Analysis and Recommendation

To open the file, it costs 1 and is executed once, and looping through all of the lines in the code costs n (the number of lines in the file), so reading the file in total costs 1 + n. To parse the file, it costs 1 \* number of lines in the file, so in total no matter what data structure used, there is at least a cost of 1 + 2n.

When using a vector, the worst case runtime of inserting into a vector is O(1), however if it does need to resize itself, it is O(n), so the overall worst-case runtime when using a vector is O(n^2). If you are simply adding the elements to the end of the vector (as we are in this case), it is very efficient; however if we did choose to insert an element in the middle of the vector, it can lead to more cost to run as it is not very efficient.

To use a HashTable, the worst runtime complexity is O(n), however on average it generally falls under O(1) to insert, search, or delete data, as long as the hash algorithm used is a good algorithm and it is not very often that the O(n) complexity happens.

When using a tree, to search the tree has a worst-case of O(n), and the time complexity is O(h) (h stands for the height of the tree). Trees can be a bit more complex to code and implement, however it does allow for the easiest and fastest search times.

When looking at the runtime complexity, and the overall data structure as a whole, I would suggest using the Binary Search Tree to house the data from the file. The BST does have a bit more complex of an implementation, but is one of the easiest ways to add or access the data and will give the best runtime complexity for the code. Since we need to use the program to not just add the lines from the file to a data structure, but also allow for easy access to read the lines in the data structure, the BST is the best option to code.