1. What data structure did you finally use for vectors? What is the asymptotic memory

usage of a vector? Of all of your vectors? Is this memory usage reasonable and why?

We used a hashmap for our vectors, they store a word as a key and then they have another hashmap as the sub-vector for that word. The asymptotic memory usage of a single vector depends on the amount of words that are in a sentence with that one word *∈O(S)*. For all of the vectors it’s the how ever many words we have times the number of words associated with that word for that word *∈O(NS)*. I think it is because we don’t store data for anything other than the words that are actually associated with a unique word. This allows us to not have a ton of zeros for the each unique word.

1. What algorithm did you finally use for cosine similarity? What is its asymptotic running

time? Is this running time reasonable and why?

Our algorithm runs through the all the words that are in the same sentence as the word that the user chooses. As it runs through the sentence it squares all the values. Then it runs through the map of the key word that the user chooses, it multiplies the values of the word map and the map of the word the program is comparing to. Then the program squares all the values of the word that is being compared to. After this the program has the numerator and to get the denominator the program takes the square root of the two values that had been squared. Finally, the program divides the numerator and denominator to get the cosine similarity. The asymptotic run time is *∈O(S^2)*. I think that this time is reasonable because depends on the amount of unique words that the word appears with.

What algorithm did you finally use for the Top-J calculation? What is its asymptotic

running time (might be in terms of J, too)? Is this running time reasonable and why?

For top-j we add the cosine similarity to a linked list that is ordered from biggest to smallest value. When the number that the user set has been reached the linked list is finished, and the program prints the values off. This algorithm runs in *∈O(J)* time. I think that this run time is reasonable, it stops after the amount the user has set has been reached.

1. What improvements did you make from your original code to make it run faster? Give

an example of your running time measurements before and after the changes. Describe

the information that informed your choices (asymptotic running time analysis,

asymptotic memory analysis, and/or profiling).

Before we fixed our code the TopJ method would take over 30 minutes to complete. This was because we added them to an ordered linked list that was N long this would work for smaller data sets, but when we got to the whole books there were many more unique words. To fix that problem we made the list as long the number that the user inputs to check similarity for. We came up with this idea by checking asymptotic memory analysis, with the unbounded list we were had *∈O(N)* just for the linked list. After the fix we get a much more reasonable asymptotic memory, the number the user puts in.