Part 4

a. We decided to use Maps for our vectors. Our map contains every word in the file as a key and the value of each word is a map of keys of every other word it occurs with and a value of how many times they occur together. Because Maps are O(1) when fastest we would say that the asymptotic memory usage of our vector for us is O(1) or just however much it takes to store a map, key, and value. So each value has a constant O(1) value associated with it. We would say that this is reasonable because there is no way that we are aware of to make this storage system faster.
b. Our cosine similarity algorithm loops through the vector of each word exactly once, gathering the required information from there. Since each vector must be looped through to gain the necessary knowledge to perform the mathematical calculation of cosine similarity we feel this is the best way to do this. As such we believe our calculation runs in O(1).
c. Calculating the top-J was a bit harder. We made a temporary HashMap to store the cosine similarity to word Q then calculated the cosine similarity for every word. After this was performed the entire map was sorted based on a comparator. As each word was sorted using this comparator the map was shown based on how many J words were necessary to be shown. This sorting of the map takes roughly O(nlogn). We feel that this is reasonable because comparison sorting is all that we have learned so far in class and O(nlogn) is the fastest form of comparison sorting.
d. We really didn't have to make any changes to make our code run faster. The requirement is that it runs in under two minutes. Even for the final and largest book it runs on our laptops in under ten seconds for a Top-J sort. Maybe we could try to optimize it but I feel like that is best left until the end of the project after everything is working. I would rather it all run in ten seconds then be broke but have the possibility of running in eight. Perhaps that is the wrong line of thinking but after we have finished the project we will expect to refactor to clean the code up.

Part 5

- Below I have pasted each of the 3 similarities run on Book 3, with their arguments and respective outputs.
- First, there are a few similarities between them. The cosine similarity and negative Euclidean distance of norms both share the same words and order for the argument given. The negative Euclidean distance runtime and the negative Euclidean distance of norms both share very similar runtimes within the 30 seconds range. Then, the cosine similarity numerical outputs and the negative Euclidean distance of norms numerical outputs share the similarity on being $-1<x<1$.
- As for differences, the cosine similarity runtime is much faster than the other two, with a time of 6 seconds as compared to those in the 30+ seconds range. Next, negative

Euclidean distance of norms only shares one word with the other two, with it being in a different position as well. Lastly, the negative Euclidean distance is much more negative than the other two, being in the upper -300 range. Our findings show that each of these similarities share something in common and different with each other.

Cosine Similarity:
```
{seem=0.8224737559012633, alwai=0.8197328720630189, like=0.8184864226226242}
BUILD SUCCESSFUL (total time: 6 seconds)
```

Argument = -f "C:\Users\home\Documents\CS-2230\project-team-38\Book3.txt" -t man,3

Negative Euclidean Distance:
```
{time=-373.73653821910426, sai=-395.86992813296644, seem= -
    399.53973519538704}
BUILD SUCCESSFUL (total time: 32 seconds)
```

Argument = -f "C:\Users\home\Documents\CS-2230\project-team-38\Book3.txt" -t man,3 -m euc

Negative Euclidean Distance of Norms:
```
{seem=-0.5958628098794667, alwai=-0.6004450481717439,
    like=-0.6025173480944478}
BUILD SUCCESSFUL (total time: 39 seconds)
```

Argument = -f "C:\Users\home\Documents\CS-2230\project-team-38\Book3.txt" -t man,3 -m eucnorm