Part 4.

1. The data structure I used for the vectors was Hashmaps. The asymptotic memory usage of on3 vector was O(S) because insertion is at a constant O(1). The memory of all vectors would be O(N^S) since in the worst case every single word would have to insert the max number of unique words. This memory usage is reasonable because the insertion is already the fastest it can be at O(1)
2. The algorithm I used for cosine similarity was to grab every single word and put it into an arrayList, and then take the words in the topj’d word’s vector and the words in the comparing vector and combined them into an array without any duplicates. From there I went through the array and did the cosine calculation for the vectors. The asymptotic memory would be O(N^S). This is because I go through every single word and in the worst case it would have to read the maximum number of unique words possible. I think this is alright because when doing topj for words it doesn’t seem to take that long, and because using the .get() method on maps is also O(1) it makes getting the information from vectors quick.
3. For my topJ calculation I sorta explained it in section B. I split used topJ to grab every single word that appeared in the text file so my memory would be O(N). After getting all of the words in the text I ran my different similarities on it would all mostly did the same. So my memory overall for topj would also be O(N^S).
4. Because I used HashMaps from the start, my program would be able to index large files and make vectors quickly already, for the largest text file my runtime was roughly 25 seconds or so. HashMaps had an insertion of O(1) and to grab information it was also O(1) so adding data and getting data from my vectors was fast.

Part 5.

1. I indexed the first book in the example. I then ran topj man 5 with cosine similarity and it printed out : **[{mere=0.5020173521113138}, {contend=0.4843805142186268}, {fieri=0.4816037947195625}, {conquer=0.4816037947195625}, {dragon=0.4816037947195625}]**. I then indexed the second book and ran the same topj and it printed out : **[{never=0.8270694598144179}, {now=0.8260067066485802}, {made=0.823169562987981}, {see=0.8214438378194772}, {feel=0.8203436006688298}]**. I then indexed the third book and ran the same topj and it printed out **: [{alwai=0.8534635992822465}, {though=0.8523358513859551}, {seem=0.8496145129805451}, {without=0.8443640836939903}, {time=0.8442275129815406}].** Through these tests it seems that topj does get better with the more data it has. As I indexed more books and added it to the vectors, the numbers got closer and closer to 1.0. The first book gave a highest similarity value of 0.5 and just adding a second book changed that the highest to 0.8. It’s also interesting to see that none of the words stayed consistent between each topj, but the numbers did increase each time. The only word that was close was seem and see.
2. Using Euclidean Distance produced the output : **[{though=-318.29388935384856}, {still=-323.9212867349103}, {sai=-328.92552348518046}, {seem=-330.6357512429652}, {thought=-331.1902776350779}].** Using Euclidean Distance between normalized vectors produced : **[{alwai=-0.5413619874312456}, {though=-0.543441162618437}, {seem=-0.548425905696388}, {without=-0.5579174066221849}, {time=-0.558162139558838}].** Again Cosine similarity outputted : **[{alwai=0.8534635992822465}, {though=0.8523358513859551}, {seem=0.8496145129805451}, {without=0.8443640836939903}, {time=0.8442275129815406}]**. An interesting observation is that between all 3 similarities Euclidean distance between norms and Cosine Similarity had exactly the same words. Some of the words did carry out through each similarity. These words were though and seem. Though appeared either first or second, and seem appeared third or fourth.