Part 4:

What data structure did you finally use for vectors? What is the asymptotic memory usage of a vector? Of all your vectors? Is this memory usage reasonable and why?

We decided to use a HashMap as our designated data structure to hold entries of the related words, and their relation count. With (N) as the number of vectors and (S) as the length of related words within those vectors, our memory usage per vector is (S). Of all our vectors, our memory usage is (N*S) We found this memory usage reasonable because at any point, to calculate TopJ or data for calculating centroids through iteration, we need to be able to pull related word info alongside their count.

What algorithm did you finally use for cosine similarity? What is its asymptotic running time? Is this running time reasonable and why?

To calculate cosine similarity, after confirming that the magnitude of vectors wasn't zero, we iterated through the main vector related words as a set using built in vector methods. For each element of that set that was related, we then multiplied it by the corresponding value stored in the comparison vector and stored it in a running sum. Afterwards, we divided that sum by the square root of the product of magnitudes.
It's asymptotic running time is (W), with W being the amount of words, including zeros, that are stored in the main vectors related words.
We found this running time reasonable, as only one operation is applied for every value in the set, and the calculation of the sum is computed after the fact.

What algorithm did you finally use for the Top-J calculation? What is its asymptotic running time (might be in terms J, too)? Is this running time reasonable and why?

To calculate the Top-J result, we first used a HashMap of String Double entries to store the relations of each comparison word to each base word. For every word vector in the database, we then checked if it held the word we were comparing for and placed it in that relation map. "getMostRelated" then chooses the top J results from that comparison word. If not enough results are chosen, we have an additional iteration loop to provide extra results with unrelated values.
It's asymptotic running time is (N*S), with N as the amount of total words stored in the database, and S as the amount of results desired.
This running time is reasonable because the map allows us to find the best results from the values as a set.

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 asymptotic memory analysis, and/or profiling.

Originally, within our index function for the database, we used to add sentences to the “all_sentences” one at a time. Afterwards, we realized the default addAll method was much faster, and improved runtime for war and peace from 82 seconds to 62. Additionally, a major feature of the index function was using an updated HashMap to keep track of what words had already been updated. This way, we didn’t have to reiterate through words when running updates. This kept our indexing method from recreating word vectors for words already indexed The running time is then U, with U being the amount or words queued to update.

Part 5

Findings of Top-J comparing indexing one text to indexing two texts:

We observed that when multiple text files where indexed the topj result differed. This is expected behavior, as the quantities stored in our semantic vectors are update with each file that is added to the database.

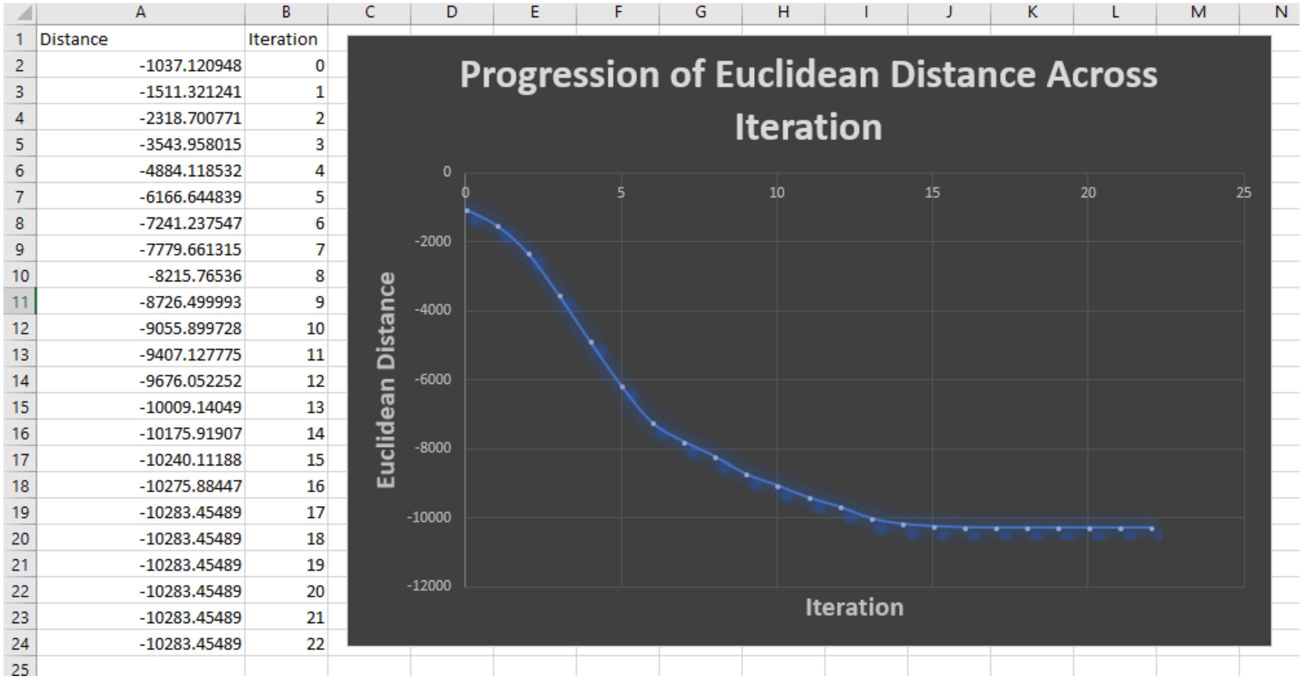
Finding on Top-J with three similarity measures:

When running Top-J with negative euclidean distance and normalized negative euclidean distance, the results were very similar. The relationships between varying words are the same in sequence, but the magnitude of difference is scaled down, due to the normalization. The cosine similarity function on the other hand presented a very similar result to that of the distance measurements. They differed slightly when out dataset became larger, but we saw consistent results.

Part 6:

Plot of average distance vs. iteration; how many iterations to converge?

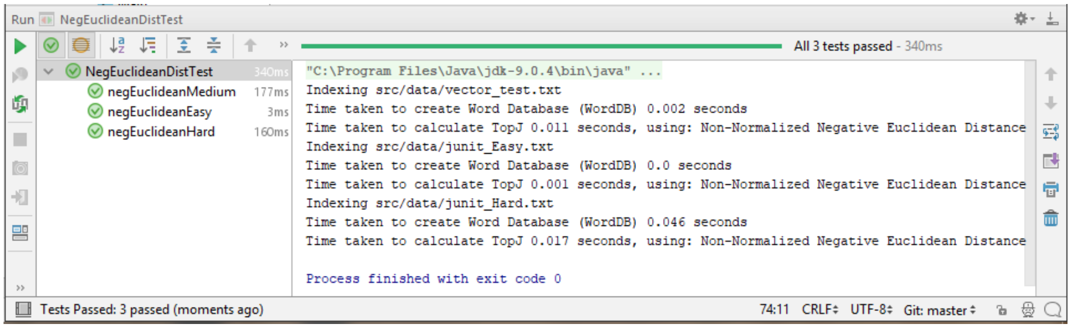
Our convergence happened after 17 iterations, at 10283.45, as shown below:



	A	B
1	Distance	Iteration
2	-1037.120948	0
3	-1511.321241	1
4	-2318.700771	2
5	-3543.958015	3
6	-4884.118532	4
7	-6166.644839	5
8	-7241.237547	6
9	-7779.661315	7
10	-8215.76536	8
11	-8726.499993	9
12	-9055.899728	10
13	-9407.127775	11
14	-9676.052252	12
15	-10009.14049	13
16	-10175.91907	14
17	-10240.11188	15
18	-10275.88447	16
19	-10283.45489	17
20	-10283.45489	18
21	-10283.45489	19
22	-10283.45489	20
23	-10283.45489	21
24	-10283.45489	22
25		

A note on our JUnit tests:

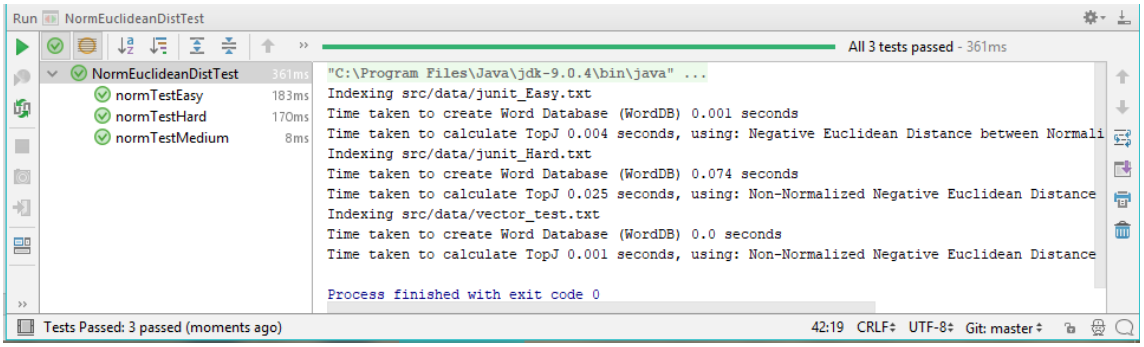
Somehow, after checking that all branches were up to date multiple times, the JUnit tests pass on my machine, a Windows, but not on Alex’s machine, a Mac. Below are screenshots of the tests passing. Any commits made past the due dates of May 4th were attempts to fix this misalignment, however, both of us have 2 slip days, so it shouldn’t matter.



Run NegEuclideanDistTest
All 3 tests passed - 340ms
NegEuclideanDistTest 340ms
negEuclideanMedium 177ms
negEuclideanEasy 3ms
negEuclideanHard 160ms
Tests Passed: 3 passed (moments ago)
74:11 CRLF UTF-8 Git: master

```
"C:\Program Files\Java\jdk-9.0.4\bin\java" ...
Indexing src/data/vector_test.txt
Time taken to create Word Database (WordDB) 0.002 seconds
Time taken to calculate TopJ 0.011 seconds, using: Non-Normalized Negative Euclidean Distance
Indexing src/data/junit_Easy.txt
Time taken to create Word Database (WordDB) 0.0 seconds
Time taken to calculate TopJ 0.001 seconds, using: Non-Normalized Negative Euclidean Distance
Indexing src/data/junit_Hard.txt
Time taken to create Word Database (WordDB) 0.046 seconds
Time taken to calculate TopJ 0.017 seconds, using: Non-Normalized Negative Euclidean Distance

Process finished with exit code 0
```



Run NormEuclideanDistTest
All 3 tests passed - 361ms
NormEuclideanDistTest 361ms
normTestEasy 183ms
normTestHard 170ms
normTestMedium 8ms
Tests Passed: 3 passed (moments ago)
42:19 CRLF UTF-8 Git: master

```
"C:\Program Files\Java\jdk-9.0.4\bin\java" ...
Indexing src/data/junit_Easy.txt
Time taken to create Word Database (WordDB) 0.001 seconds
Time taken to calculate TopJ 0.004 seconds, using: Negative Euclidean Distance between Normali
Indexing src/data/junit_Hard.txt
Time taken to create Word Database (WordDB) 0.074 seconds
Time taken to calculate TopJ 0.025 seconds, using: Non-Normalized Negative Euclidean Distance
Indexing src/data/vector_test.txt
Time taken to create Word Database (WordDB) 0.0 seconds
Time taken to calculate TopJ 0.001 seconds, using: Non-Normalized Negative Euclidean Distance

Process finished with exit code 0
```