a) 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 finally used a hashmap for the vectors that stored a word (key) and the number of times it occurs in the same sentence as the base word (value). The asymptotic memory of the hashmap for one vector of one word is O(n) (although that would be only if the word of our vector was in every sentence). The memory usage of all our vectors is O(n^2) because there are n vectors each with O(n) memory.

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

For this, what we decided to do was we decided to only for the same keys to multiple the u and the v. This is because if there is similar matching that atleast one of them has a 0, then you will be adding a zero. This is how we found our numerator. To find our denominator, we decided to make a method and will go through the entry set to square all the values and add them up. This method is then called no both of the HashMap we for the two words that we are comparing. Finally, if either u or v in the bottom is a 0, then we will return 0 and if it is not well will follow the equation where we take the numerator divide by the magnitude of each of the values in the denominator.

Well we three for loops but neither of them are nested so we came to the conclusion that it is going to be O(3N) = O(N).

Yes, because we have to go through every element and compare because of the summation.

c) 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 each word in the entire text, we make a Pair<String, Double> that holds the word and its semantic descriptor. We find the cosine similarity of this word to the base word and add the pair to an ArrayList. This ArrayList is then sorted with Collections.sort and we take the top j of them to display to the user. The sort function is O(nlogn). However, the finding the cosine similarity of each word I believe is $O(n^2)$, so this is a $O(n^2)$ runtime.

d) 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).

We changed from using ArrayLists to Hashmaps as our base data structure for vectors. For example, I tried running the top-5 of a word on the Swann Way text. Initially, this ran for over 20

minutes before I shut it down. After implementing hashmaps, it ran in under 2.5 minutes. This change was made because hashmaps have a lookup time of O(1) rather than the O(n) in arrayLists. This allows us to check if a word is in a vector or not very quickly.