**CS 2302 Data Structures**

**Fall 2019**

**Lab Report #4**

Due: October 21, 2019

Professor: Olac Fuentes

TA: Anindita Nath

**Introduction**

For this lab, I implemented different functions using B-trees and Binary Trees. These functions were used in order to find the similarity between a pair of words. The purpose of this lab was to implement B-trees and Binary trees in a way that compared two words to each other by their word-embeddings which consists of 50 floating point numbers. The similarity was then measured from very different to identical. Search algorithms, and Insert algorithms were used for both types of trees to aid in finding these similarities.

**Proposed Solution Design and Implementation**

**B-Tree Implementation:** For the B-tree implementation I began by creating a empty B-Tree. I then continued by reading the glove file that contains all the words and embeddings. While the file was read, I added each word and embedding to the B-tree. I used an insert functions that inserts items into a B-tree and sorts the items in alphabetical order. After the B-tree was created, I then read the file that contained the many pairs of words. Each word was searched for in the B-tree. The similarity between the pairs were then computed by using the cosine distance which ranges from -1 to 1, -1 being very different, and 1 being very similar. While all of this was being done, I had a timer running to calculate the times required to build the B-tree and to compute the similarities.

**Binary Tree Implementation:** For the Binary Tree Implementation, I first created an empty Binary tree, then added objects to it. The objects were created while reading the text file that contained words along with their embeddings. I inserted these objects, then had the program read the file that contains pairs of words. Each word from each pair was then searched for within the binary tree using a search algorithm that searches based on the word in each object. Once it was found, the embeddings were then used to compute the similarity between the two words. The similarity was calculated using the cosine distances between the two words. Meanwhile, a timer was running for the construction of the Binary tree, and while the word was being searched and the cosine distance was being calculated.

**Experimental Results**

To test the program, I tested each function with different text files. The first text file contained 15 pairs of different words, the second one contained 50 pairs, and the last one contained 100 pairs of words. The efficiency of the two methods was found by timing the construction of the trees, and the query processing where the searches for the words were made. I compared the running times of insertion for both types of trees, and the search method for both trees.

**Test #1:**

The first test consisted of a text file with 15 different pairs of words. The max data for the B-tree is 3. The construction of the Binary tree was faster, and for the query processing the Binary tree implementation was also faster.

**Test #2:**

The second test consisted of a text file with 50 different pairs of words. The max data was changed to 4. The Binary tree construction was once again the most efficient, however for query processing, the B-tree implementation was more efficient.

**Test #3:**

The third test consisted of a text file with 100 different pairs of words. The max data was changed to 3 for the B-tree. The Binary tree was most efficient while creating the tree, and the running times for query processing were very close in time, and were both efficient times.

**Running Times for creation of Tree:**

|  |  |  |
| --- | --- | --- |
| **Test** | **Binary Tree** | **B-Tree** |
| 15 pairs of words | 17.191797 | 47.33265 |
| 50 pairs of words | 34.645288 | 46.84082 |
| 100 pairs of words | 17.216610 | 20.61792 |



**Running Times for query processing:**

|  |  |  |
| --- | --- | --- |
| **Test** | **Binary Tree** | **B-Tree** |
| 15 pairs of words | 0.0151245 | 0.019942 |
| 50 pairs of words | 0.0167689 | 0.032287 |
| 100 pairs of words | 0.0550267 | 0.057820 |

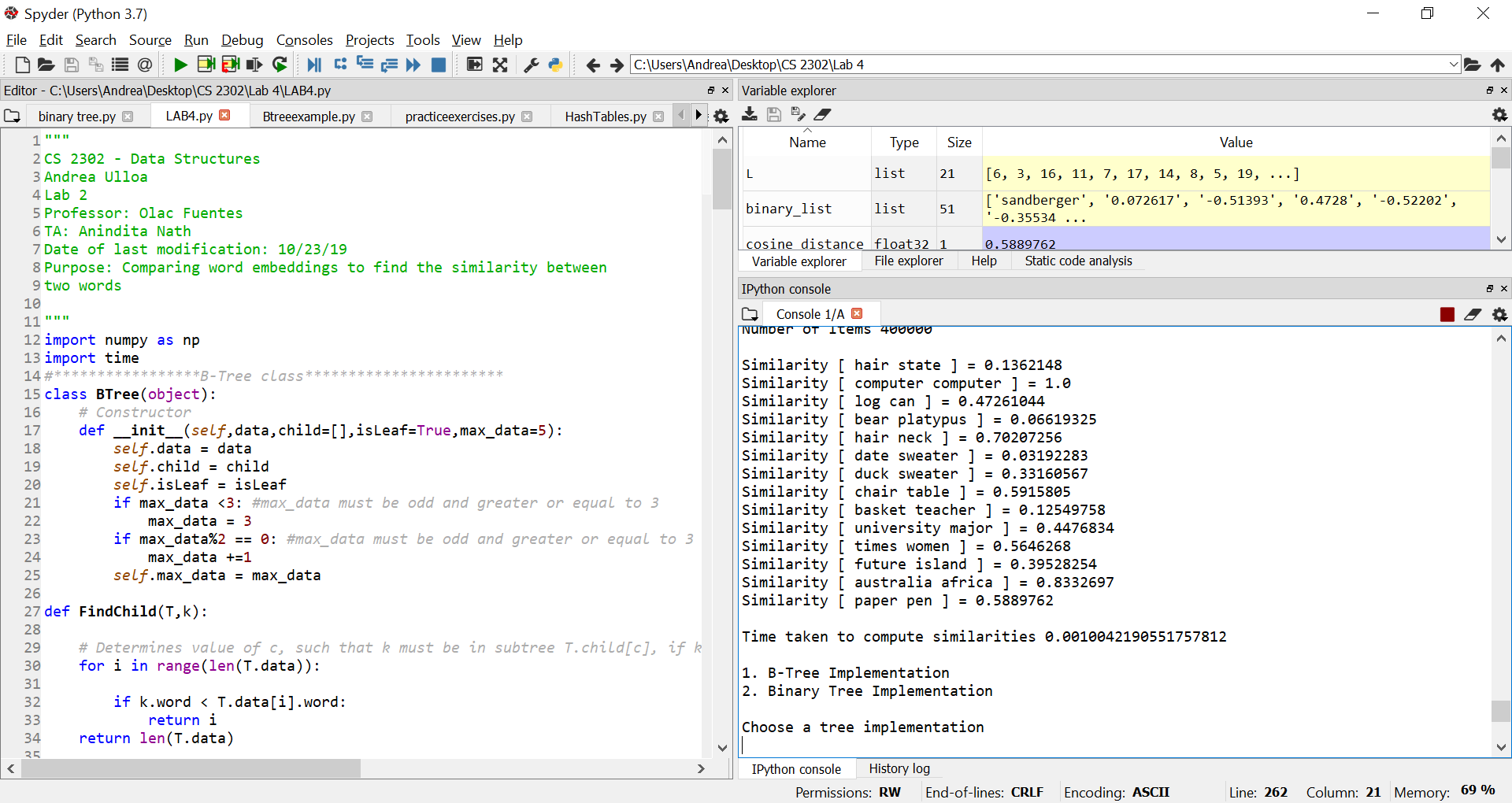


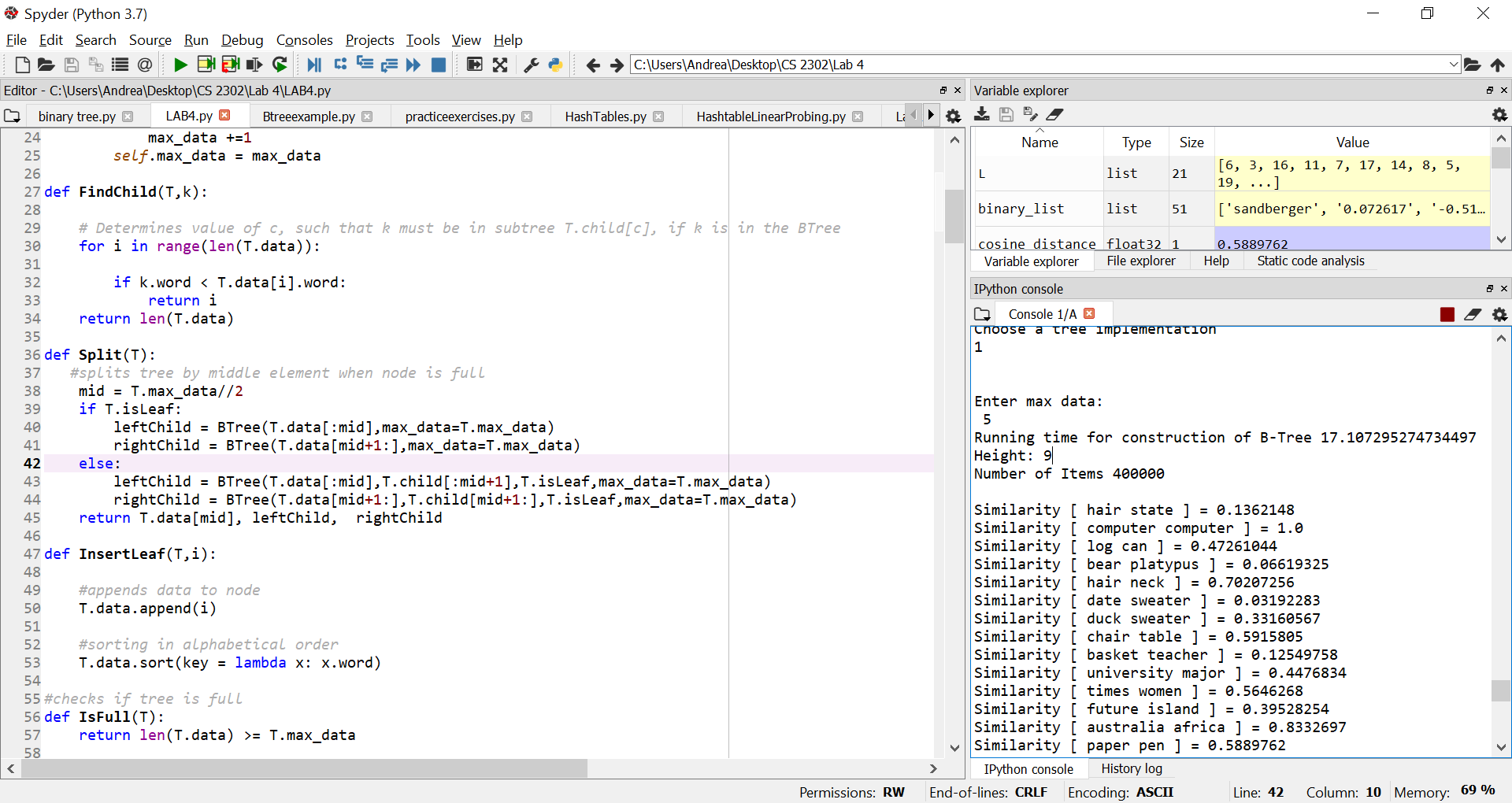
For the most part, the construction of the binary tree was the most efficient which makes sense since there is no splitting of nodes that is being done, such as in the B-tree implementation. For the searching of words, the B-tree came out to be the most efficient, however the Binary tree search implementation was not far behind. They both are efficient when it comes to searching since you don’t have to search the entire tree in order to find an item within it.

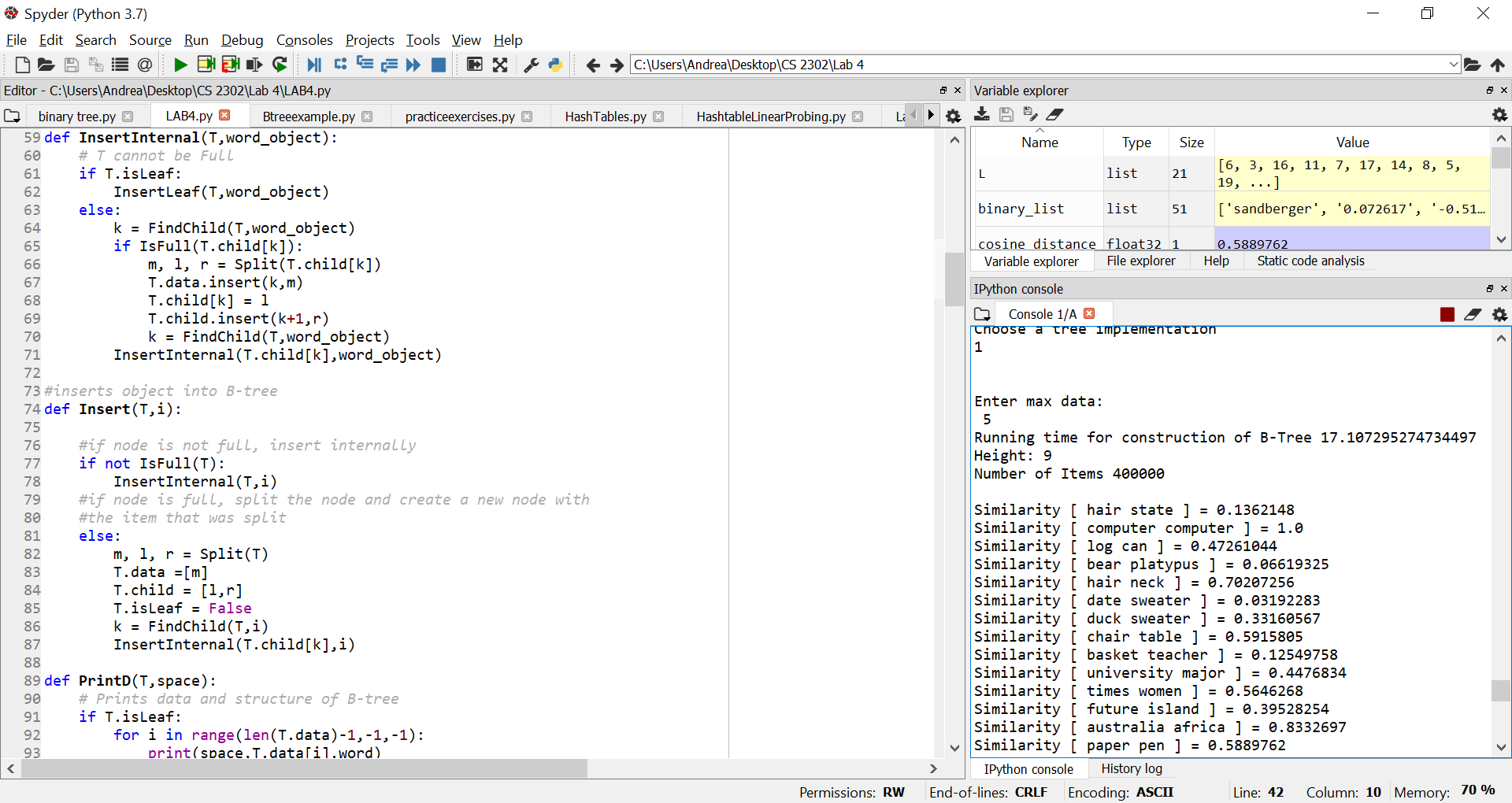
**Conclusion**

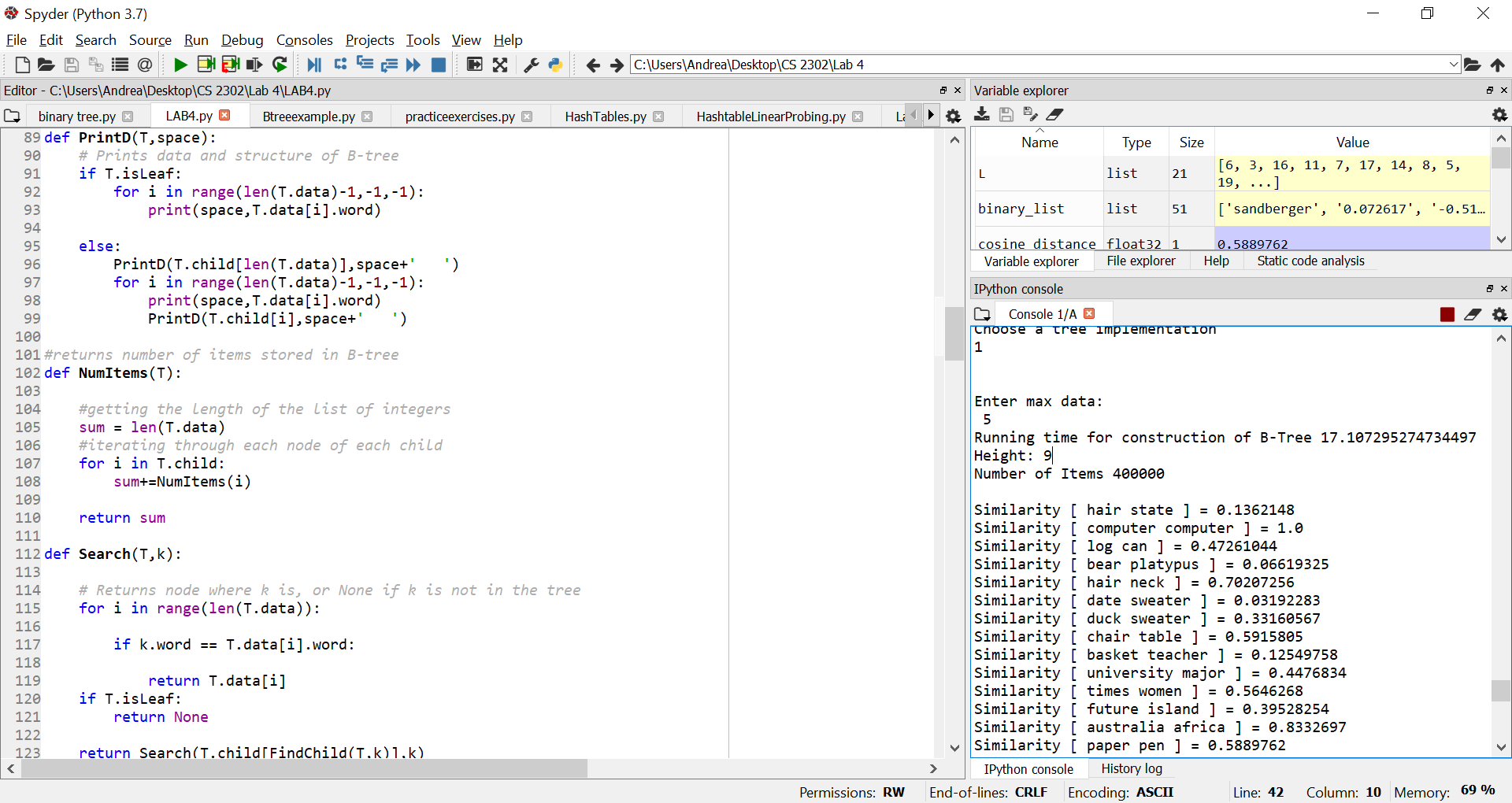
In this lab I learned more about how to implement B-trees and Binary Trees. I also learned about the efficiency of each type of tree while performing different operations. Both types of trees perform differently when having to search for different words. Increasing inputs determine the overall efficiency of the insertion and search functions for both trees. Overall, the most efficient implementation for insertion was the Binary tree implementation. When it came to searching, B-tree was more efficient in most cases, however not far ahead of the Binary tree running time. Another aspect of this lab that I learned more about were the different word embeddings that every word has. I learned that making comparisons of words is done by comparing the word embeddings by using the cosine distance.

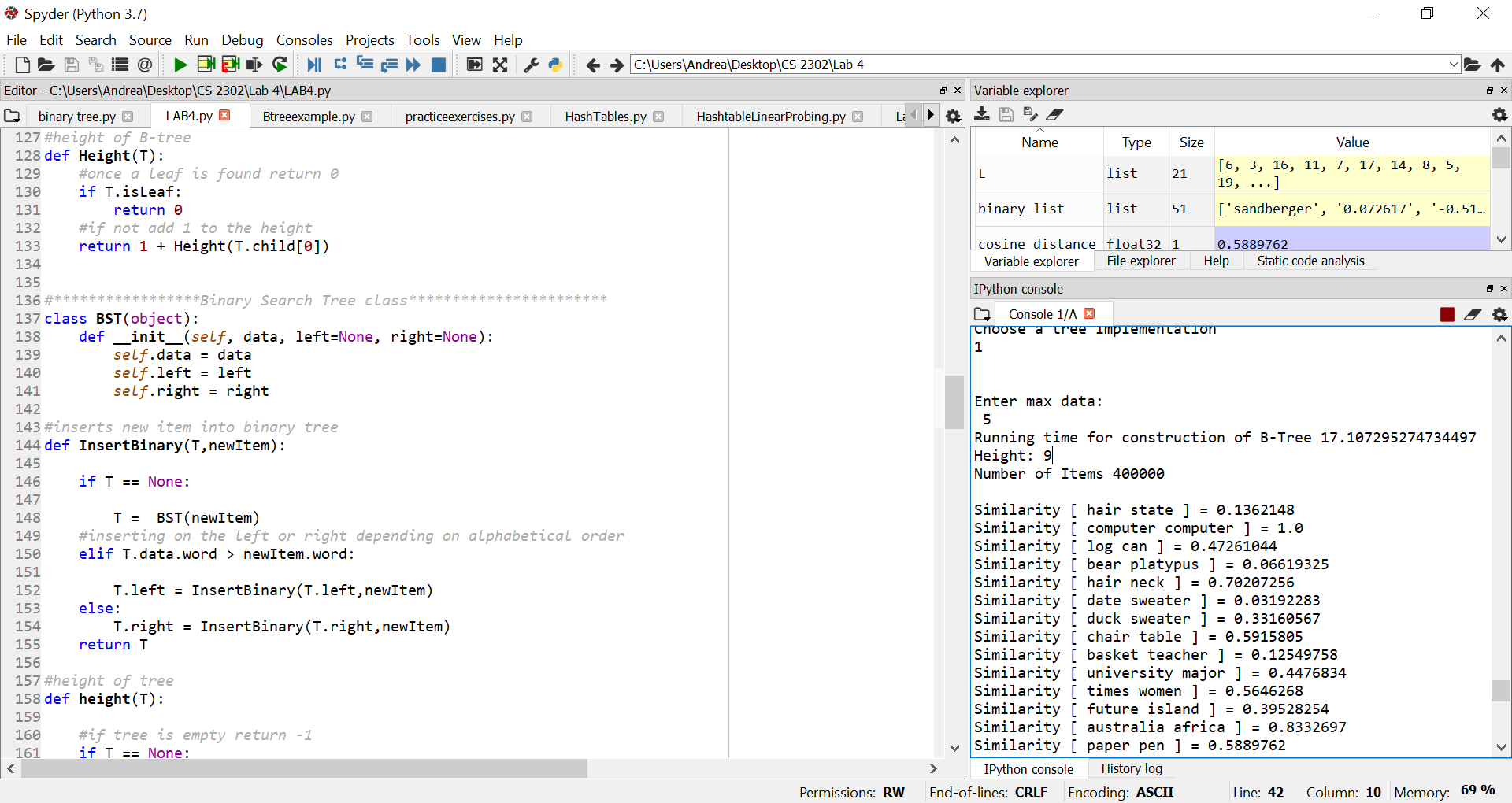
**Appendix**

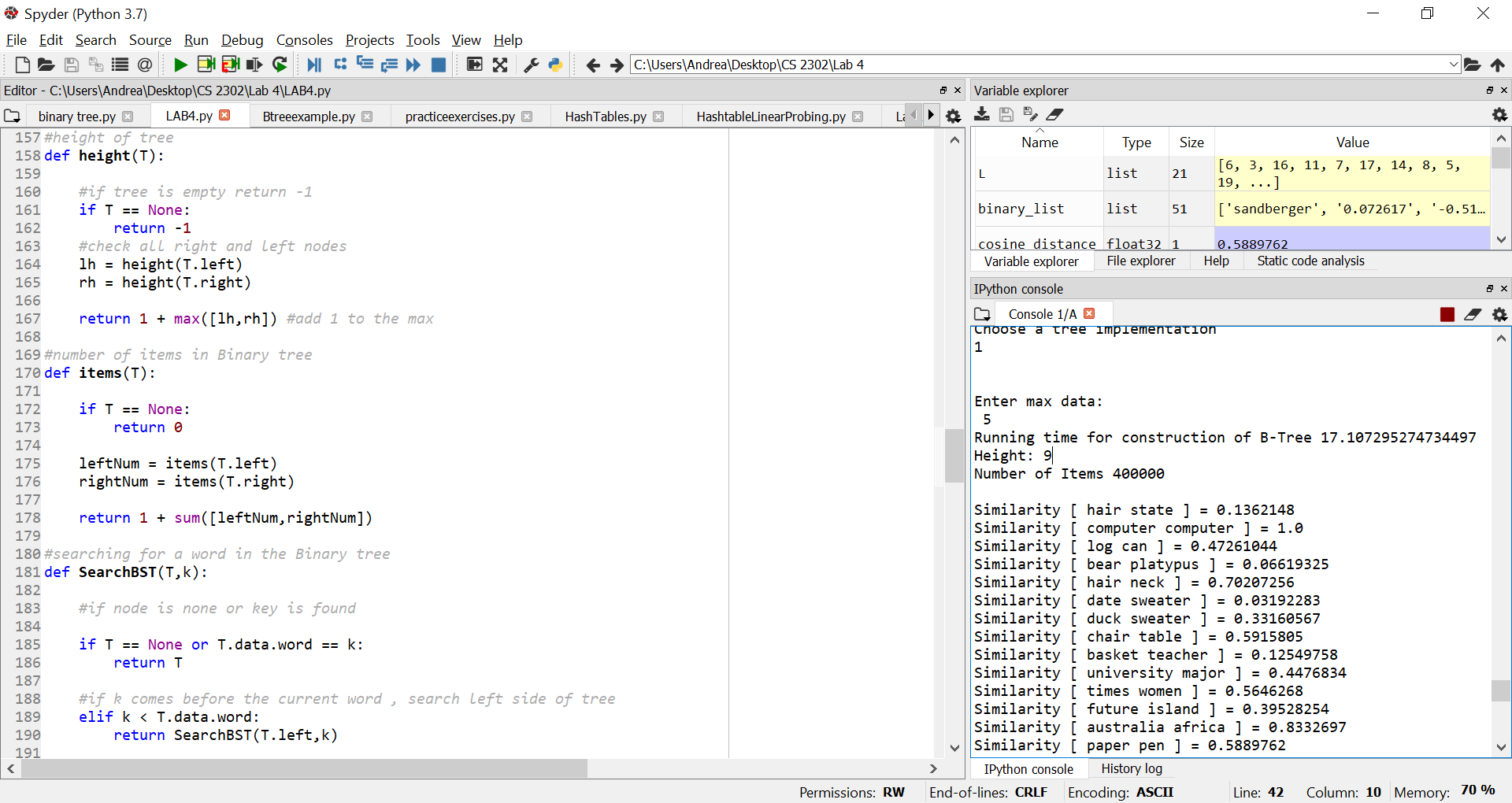


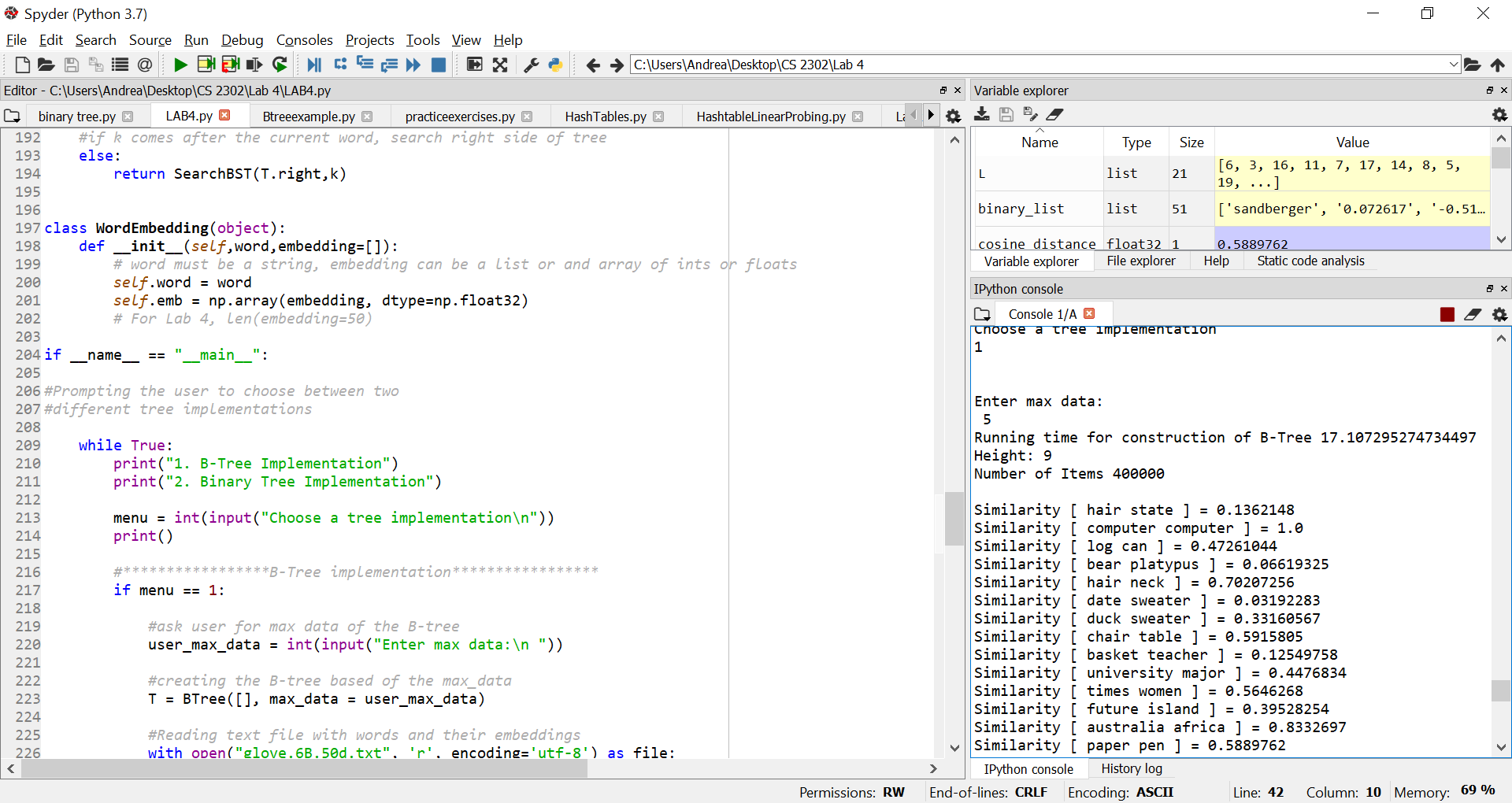


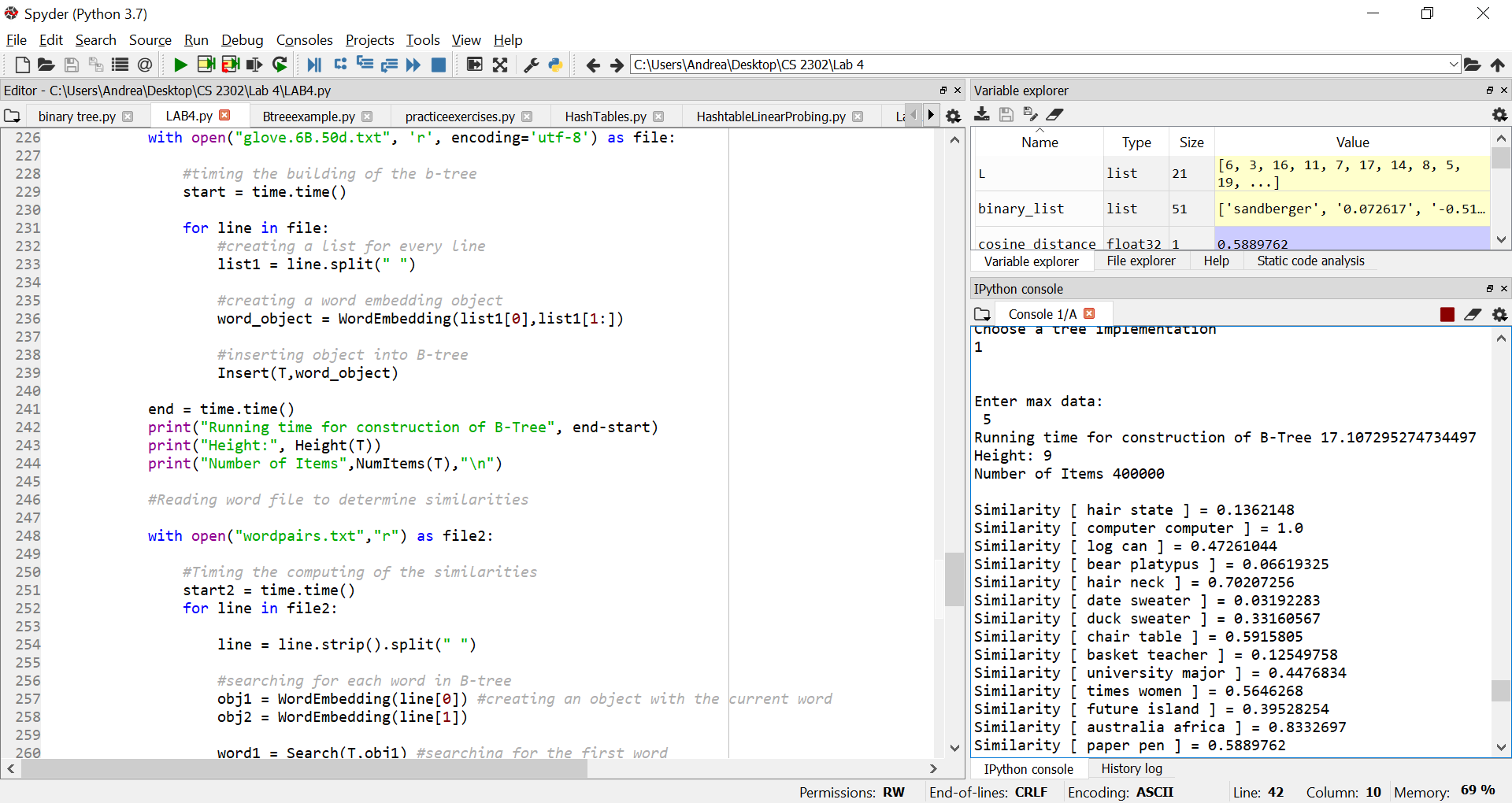


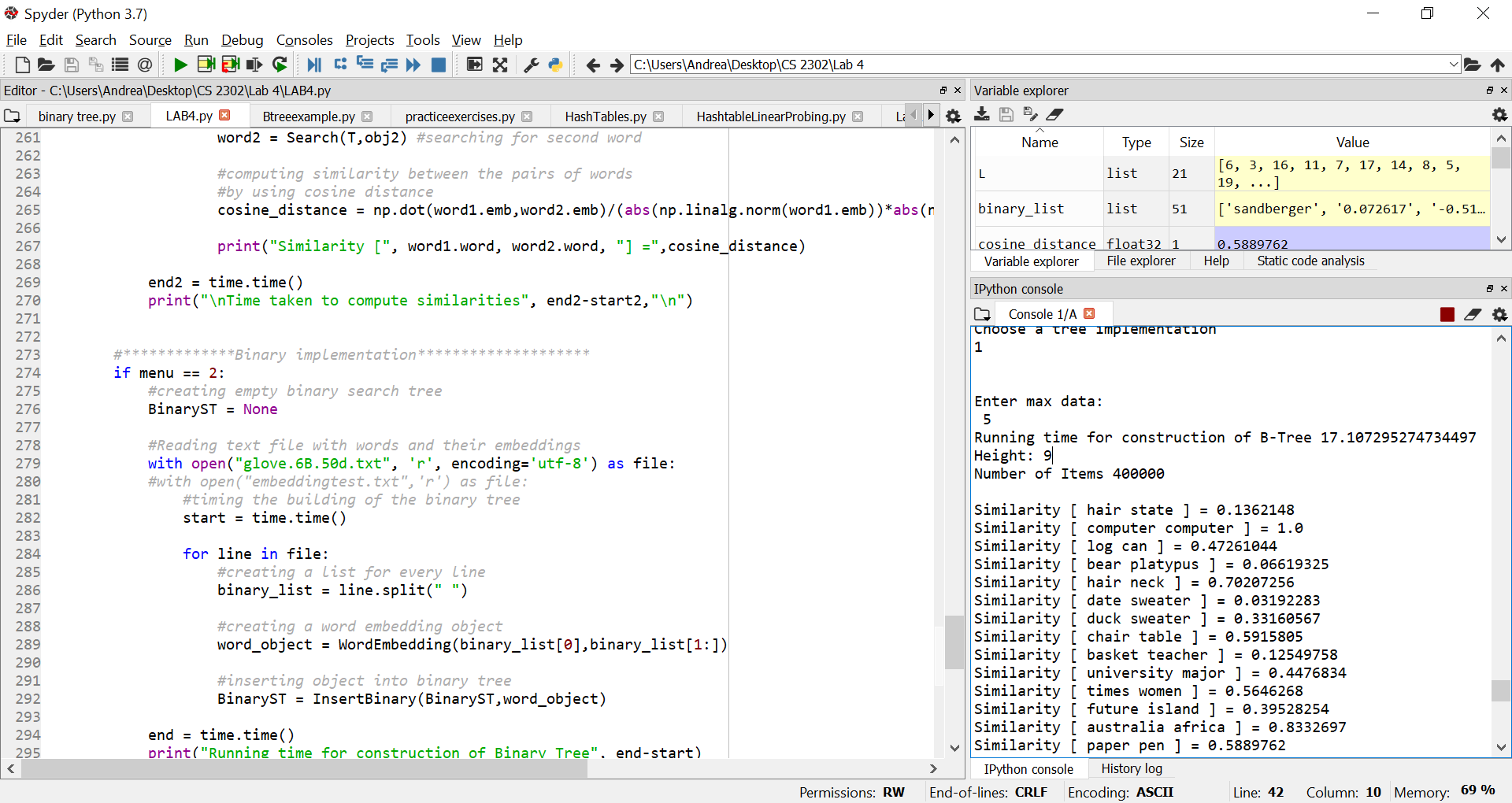


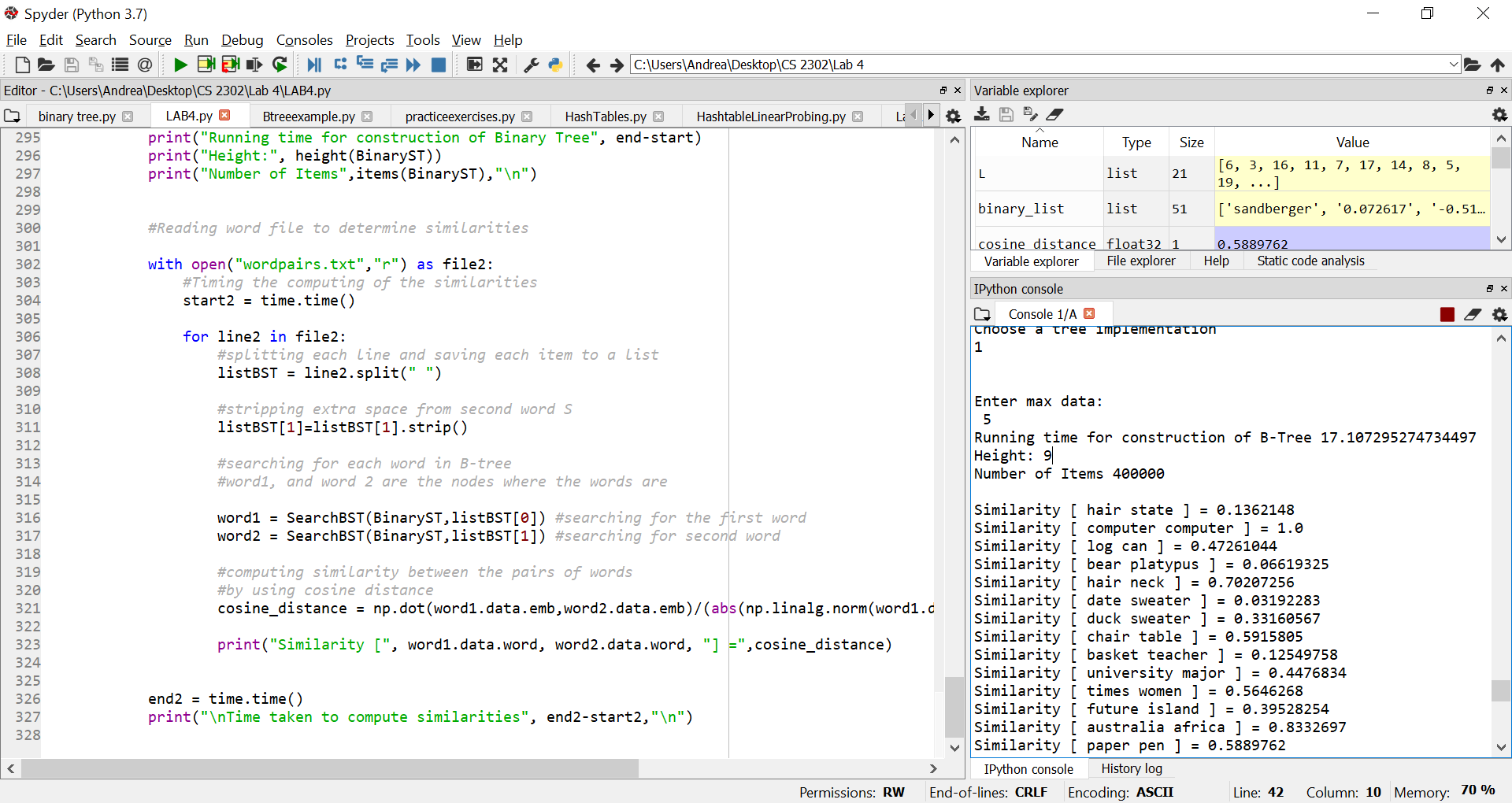












I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class