**Problem Description**:

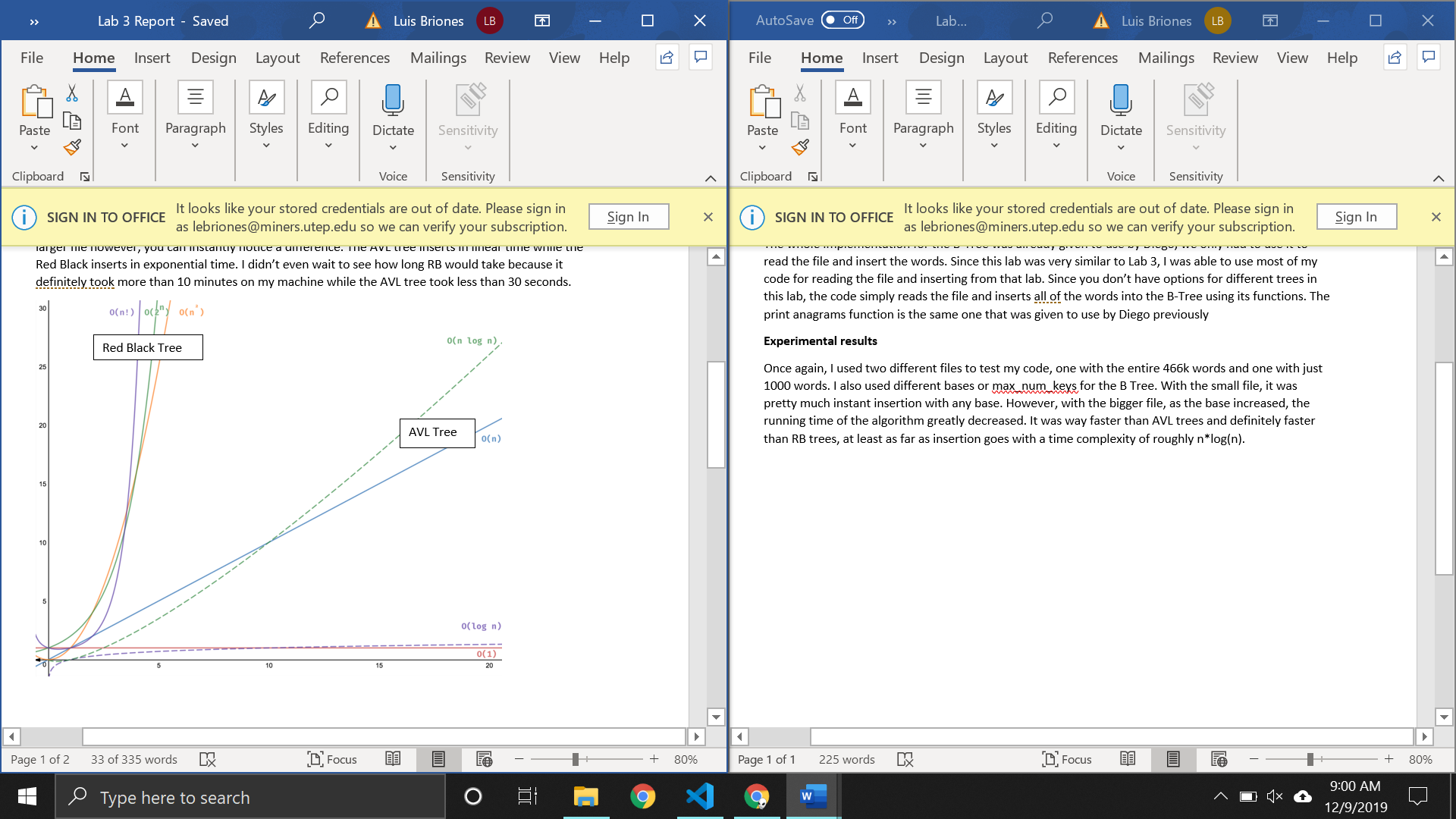
We have to read a file of over 466k thousand words and create a data structure called english\_words using a B-Tree that stores all of the words.

**Solution design and implementation**

The whole implementation for the B-Tree was already given to use by Diego, we only had to use it to read the file and insert the words. Since this lab was very similar to Lab 3, I was able to use most of my code for reading the file and inserting from that lab. Since you don’t have options for different trees in this lab, the code simply reads the file and inserts all of the words into the B-Tree using its functions. The print anagrams function is the same one that was given to use by Diego previously

**Experimental results**

Once again, I used two different files to test my code, one with the entire 466k words and one with just 1000 words. I also used different bases or max\_num\_keys for the B Tree. With the small file, it was pretty much instant insertion with any base. However, with the bigger file, as the base increased, the running time of the algorithm greatly decreased. It was way faster than AVL trees and definitely faster than RB trees, at least as far as insertion goes with a time complexity of roughly n\*log(n).



B-Tree

**Conclusion**

Overall, B-Trees are faster than AVL and Red Black trees when it comes to insertion by far. Between lab 3 and 4 I learned how to compare running times of different data structures and how different factors affect them.

**Source code**

<https://github.com/lebriones/CS2302/blob/master/Lab4.py>