CS2302 Data Structures

Fall 2019

Lab Report #2

Due September 20th, 2019

Professor: Olac Fuentes

TA: Anindita Nath

**Introduction**

For this assignment we has to implement recursive versions of Bubble sort and Quick sort. We then had to modify our Quick sort methods to only sort the sublist where the kth element is located. Then we have to implement a Quick sort method using only a stack instead of using recursion, similar to the Tower of Hanoi example. Finally we had to implement a Quick sort using a while loop.

**Proposed Solution Design and Implementation**

Bubble sort: For bubble sort I will use a traditional bubble sorting method that will use a loop to make swaps and use a counter in the header to be able to reach a base case.

Quick sort: For the first quick sort method I will be using a traditional quick sort method. I will use a separate partition method that will help us sort elements to the right and left of the pivot. For the pivot I will simply choose that last element and the middle element since the lists will be either ordered or reversed. By choosing the last element I will ensure that I will choose either the max or min element as the pivot, which will give us our worst case time complexity. When I choose the middle element as the pivot I will ensure that I choose the best case time complexity.

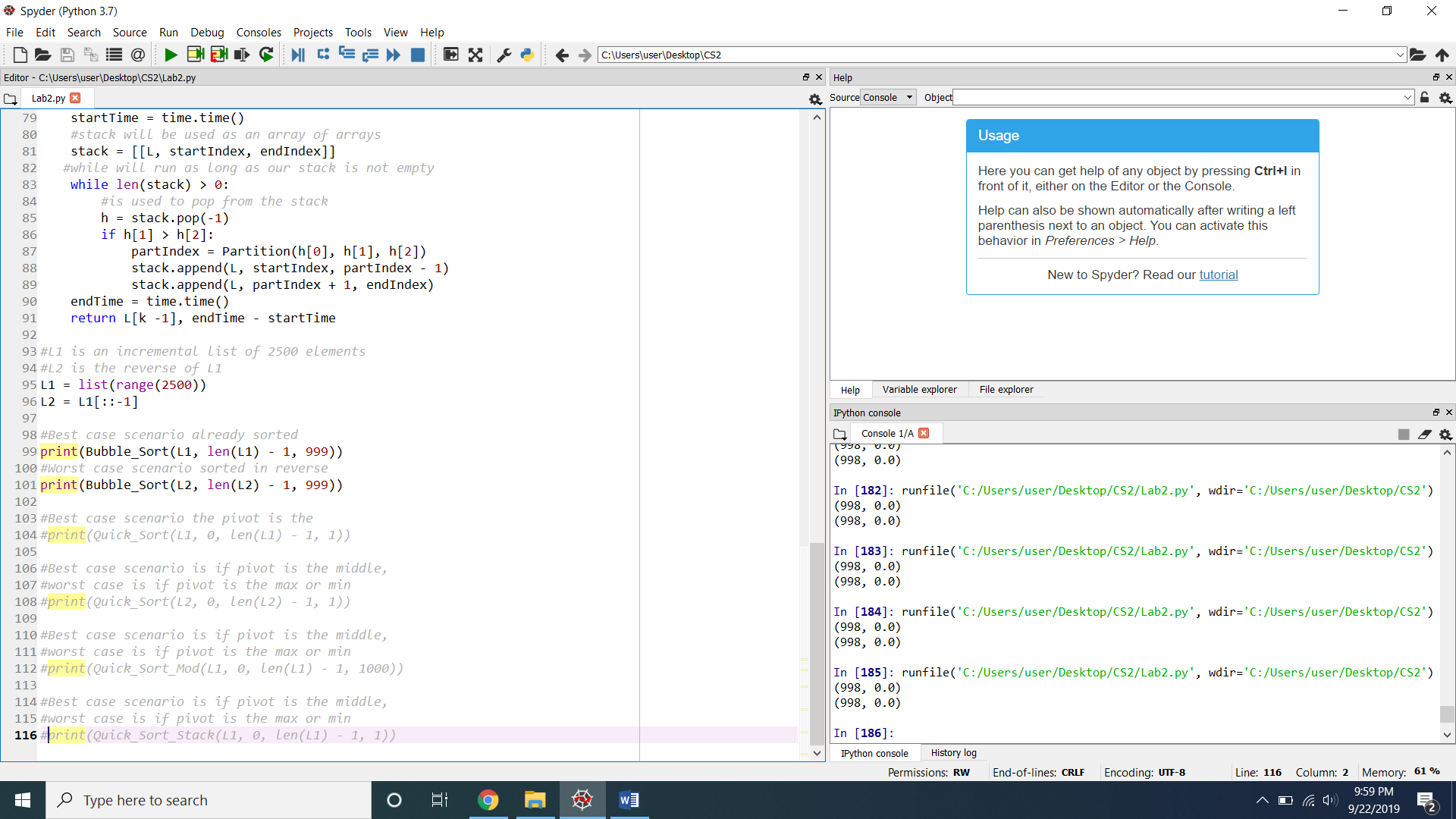
Quick sort modified:

Quick sort stack:

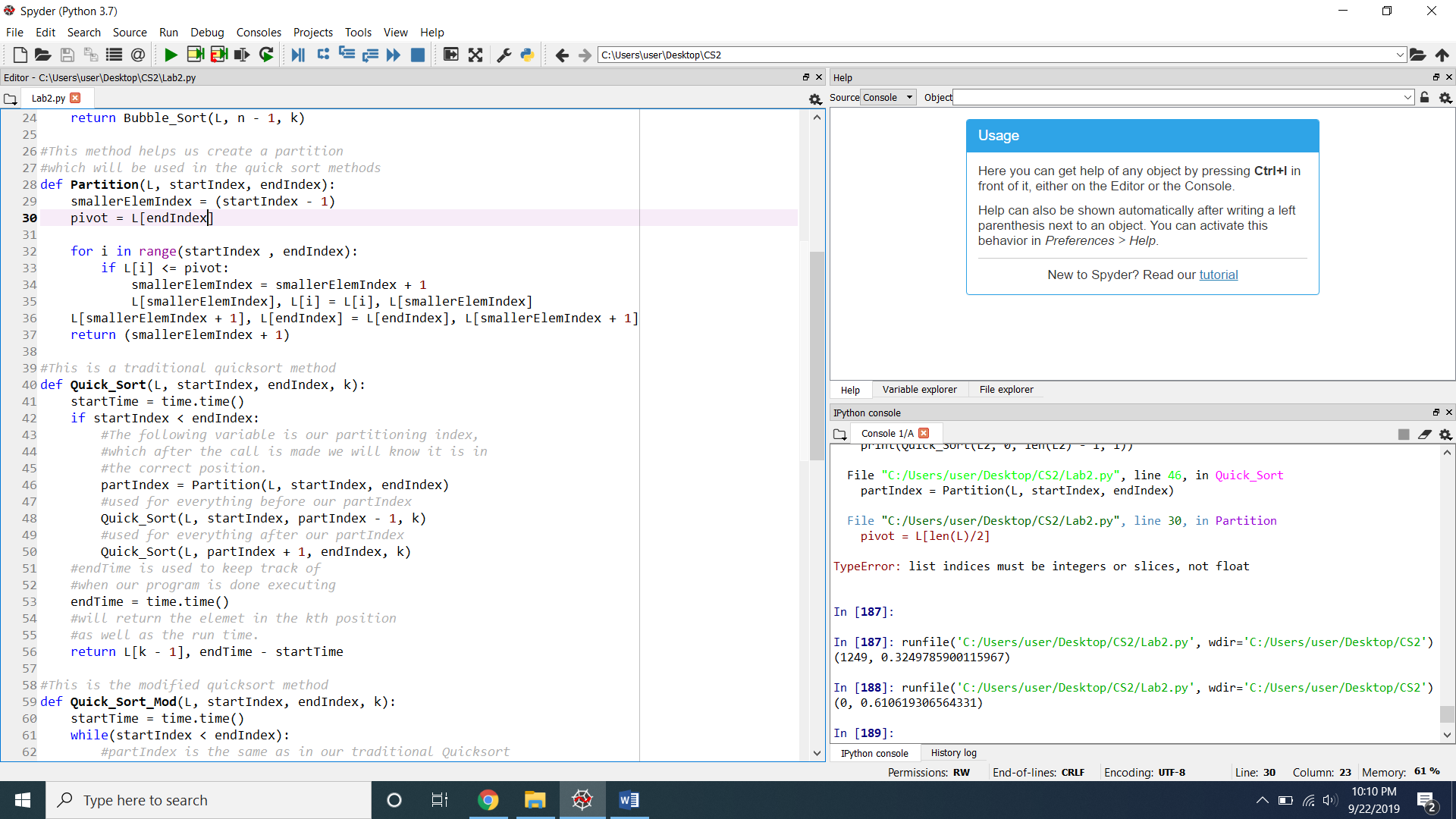
Quick sort while loop: I was unable to implement this version of quick sort

**Experimental Results**

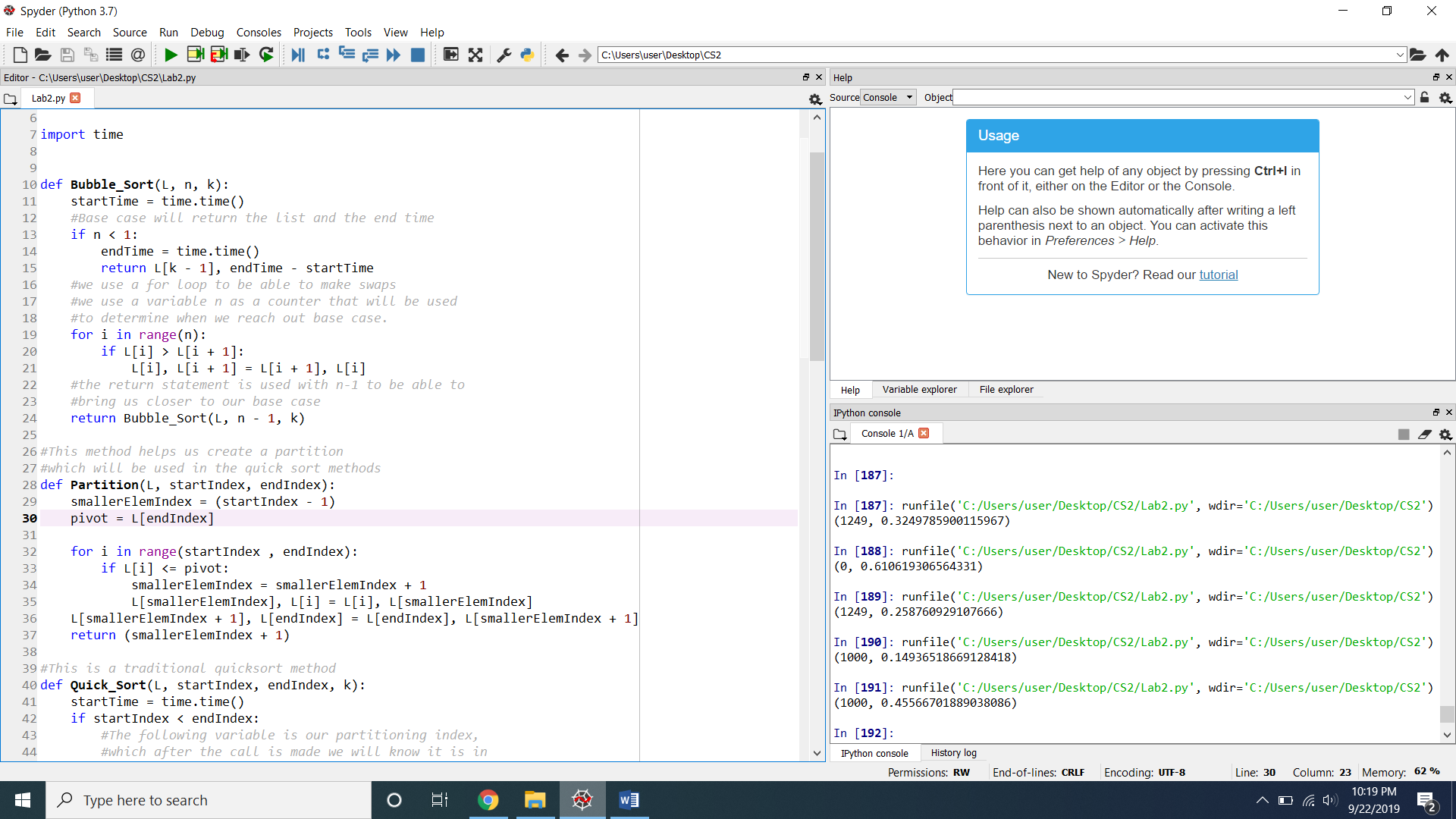
Bubble sort: I first printed the results of the sorted then reversed sorted lists to see the differences in time complexities. The image displays my results, however I was unable to get a time complexity. Each return first shows the kth elements and then shows the total execution time. My theoretical results would assume that a sorted list would be executed quicker than an reversed sorted list as there would not be any swaps:



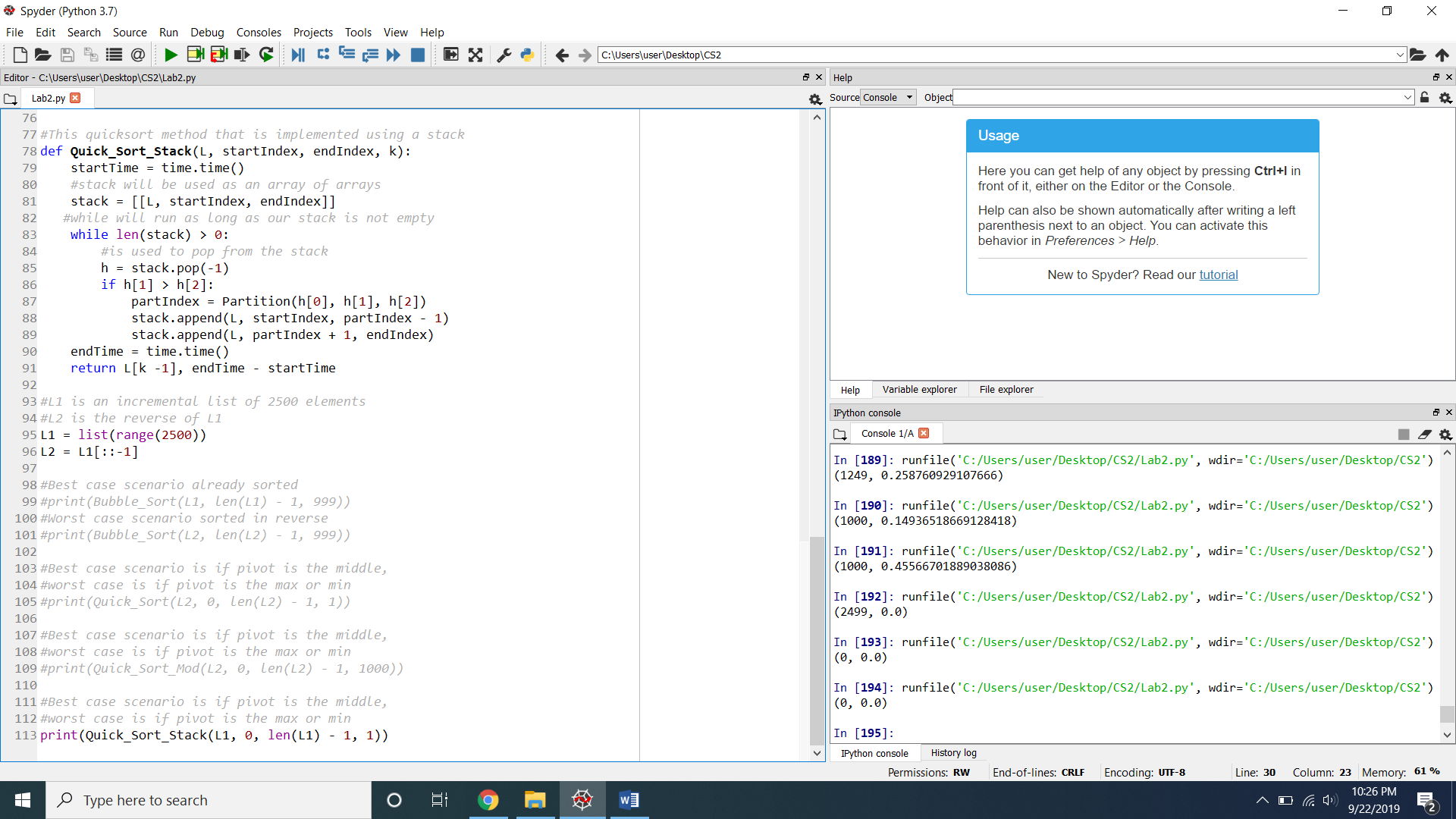
Quicksort: I used a reversed sorted array for all of my quick sort methods. I first implement a quick sort method where I chose the middle element as the pivot for the best time complexity as seen in run 187, then the second run I changed the pivot to be the last element so it would be a min variable so that it would produce the worst time complexity as seen in run 188. Each run has first returns the kth element then returns the execution time. Our experimental results are as what was expected with our theoretical results



Quick sort Mod: I applied the same procedures as I used for the quick sort method the best case is shown in run 190, and the worst case is shown in run 191. Our experimental results are as what was expected with our theoretical results:



Quick sort stack: I applied the same procedures as I used for the quick sort method the best case is shown in run 193, and the worst case is shown in run 194. I was not able to get any execution times for the methods, which did not allow me to get any experimental results. :

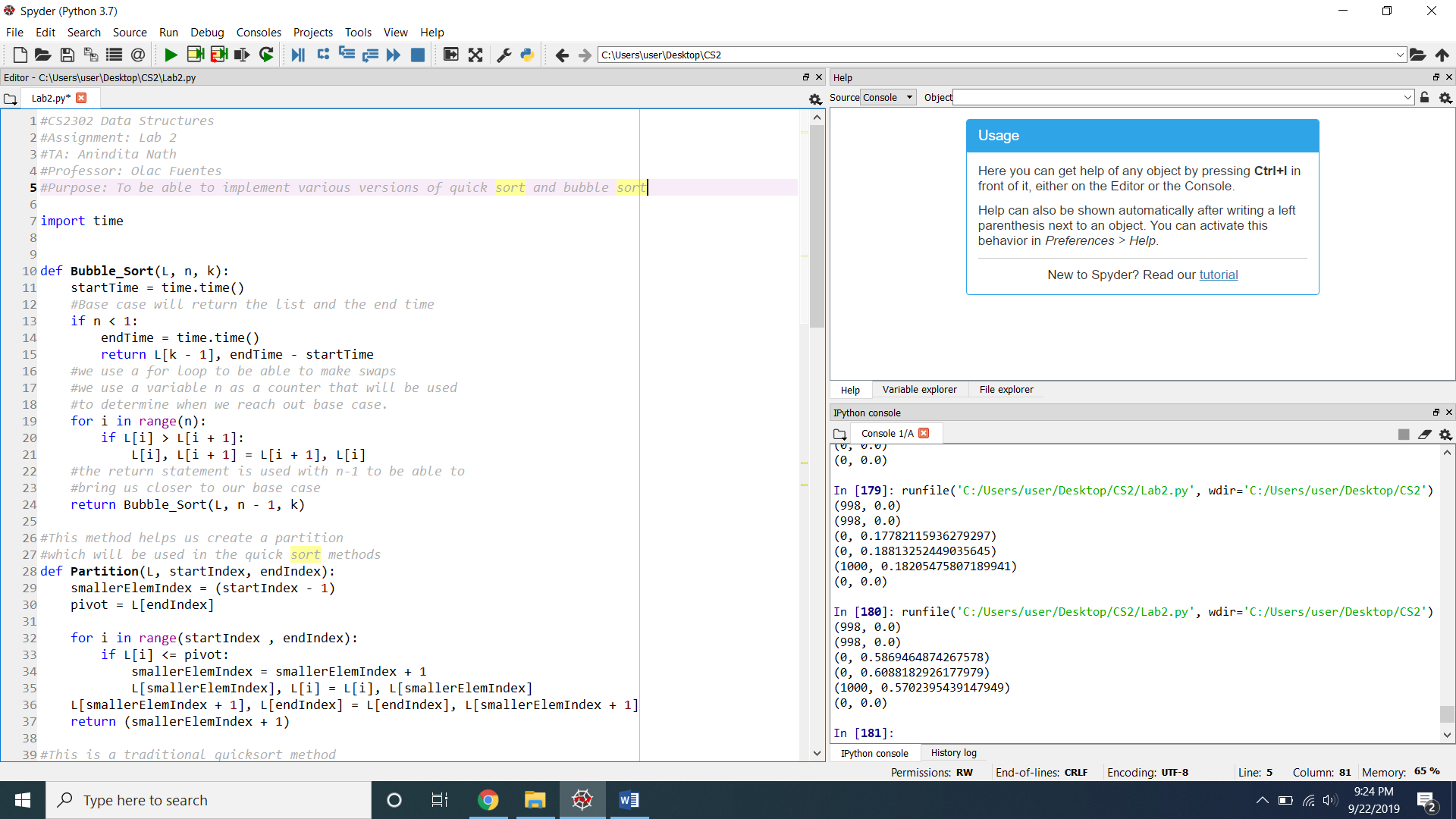


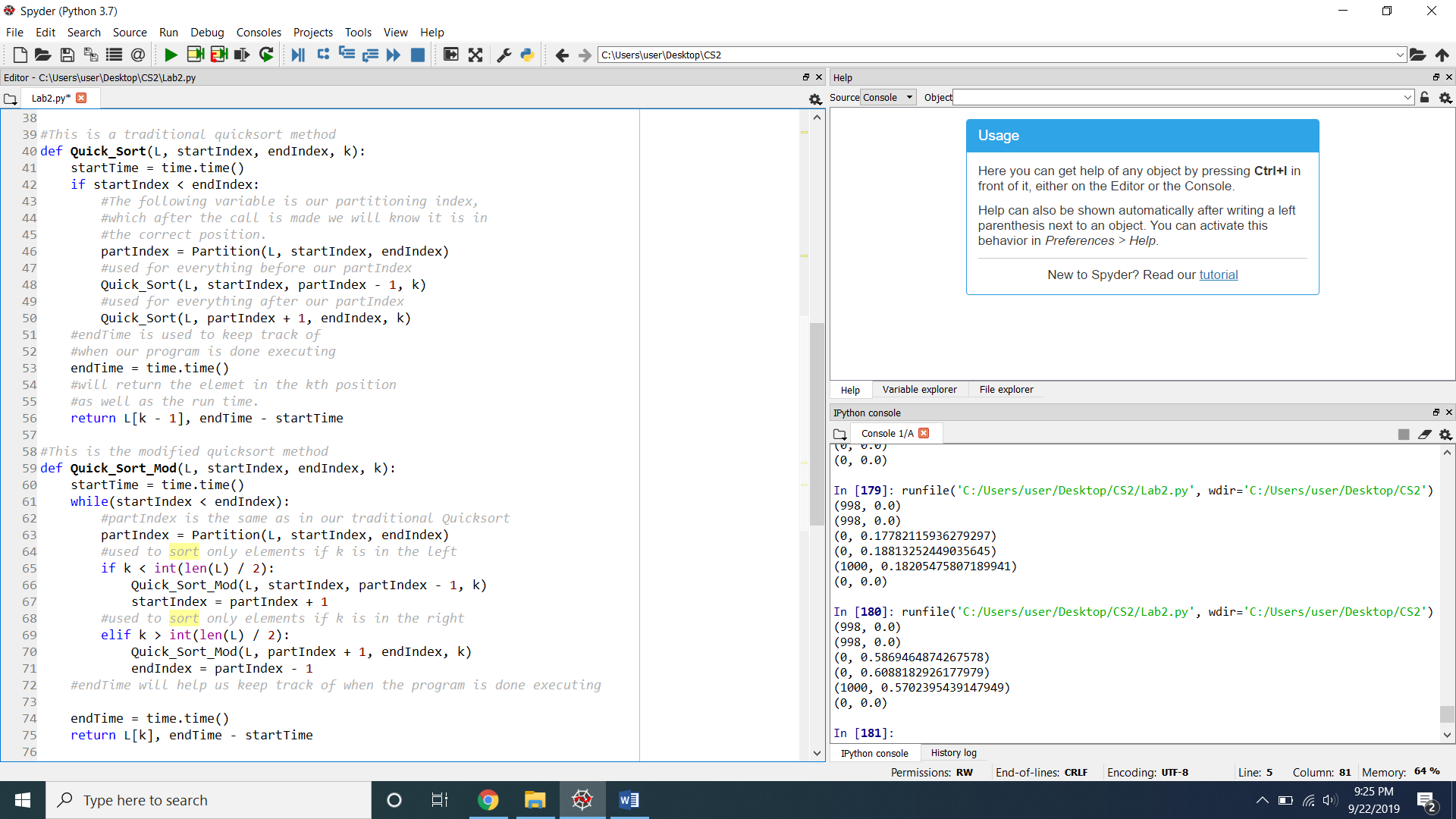
**Conclusion**

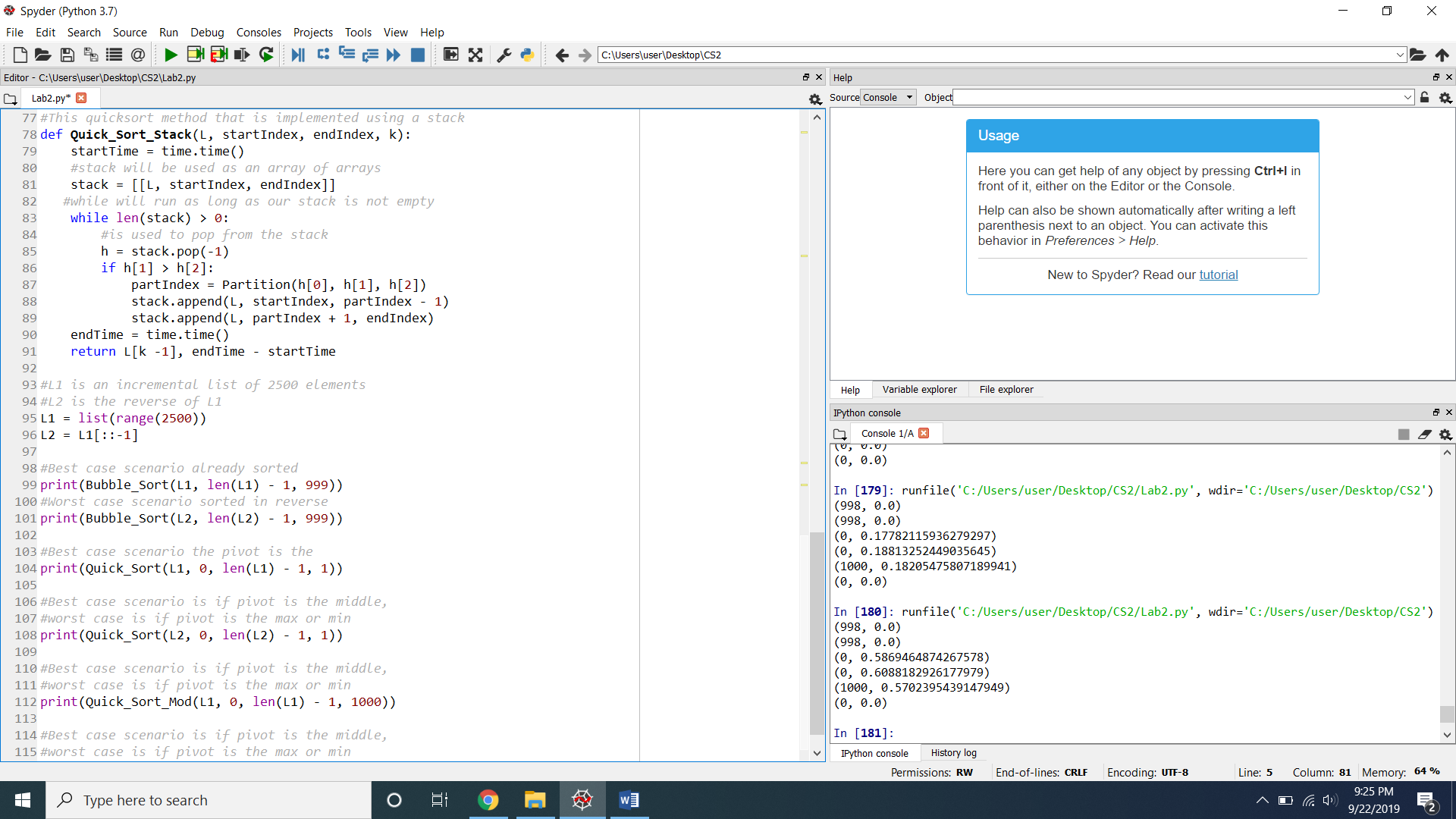
This lab helped me understand the time complexities of different sorting algorithms. I was only used to making these sorting algorithms using iteration, however this lab helped me learn how implement these algorithms using recursion. This lab was challenging for me and took me a long time to implement, especially part 2 of the lab. I was able to see that our experimental results coincided with what was expected from based on our theoretical results. I also learned how to decide which tests I should run for each method to get the best and worst case scenarios.

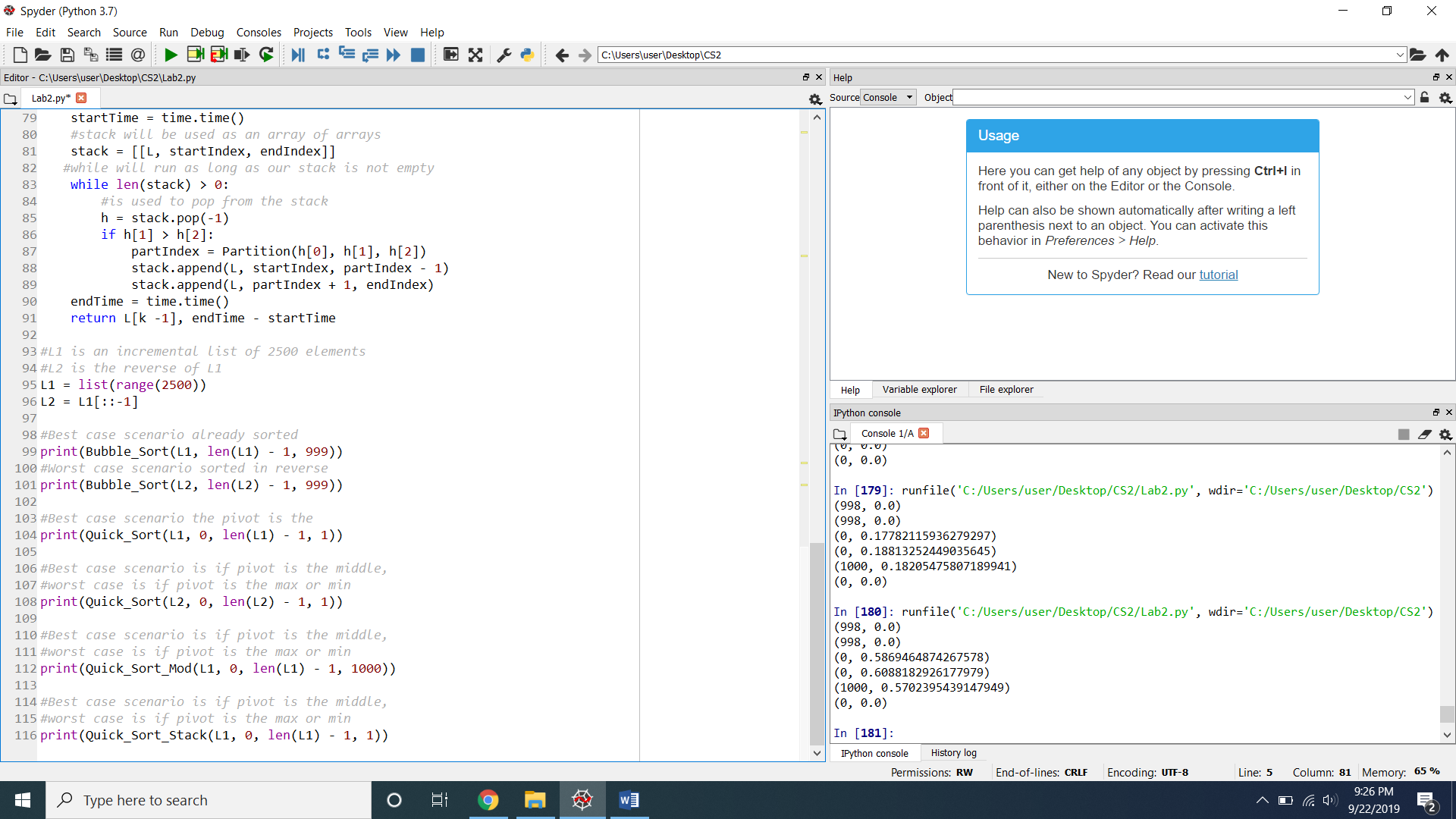
**Appendix**

The following is my source code









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