CS 2302 - Lab 2

Sofia Gutierrez

Olac Fuentes

**Introduction**

The objective of lab 3 was to create a program to find the kth smallest value in a list. Both k and the elements in the list were taken from the user. The program contains a total of five methods all of which showcase the kth smallest element in the list. These methods included Bubble Sort, Quicksort, and modified versions of Quicksort. Four out of five of these methods included their own partition method. Each partition method would take the user’s list which was then divided and sorted so that all the elements less than the chosen pivot were to the left of it and all the elements greater than the pivot were to the right of it. Since it was not necessary to find and print all the k values for each of the methods- especially since if done correctly, all the k values would be the same- the user would select the desired sorting method that would find k. The time it took for the method to execute was printed alongside the value of k.

**Design and Implementation**

The design of this program started off by storing the length of the list that was desired by the user as well as the elements within the actual list. The user would also have control of the kth smallest element in the list they wanted to print out. Next came the layout of all the methods that would sort the user’s list so that it would be possible to find the kth value. Bubble Sort was the first method that was displayed and was set up to have a for loop where the variable ‘i’ iterated through the length of the list. Nested in was a second for loop where the variable ‘j’ iterated through the length minus the ith variable since those last elements would already be sorted. Bubble sort then compares the current element where the ‘j’ variable is currently at to the next element that follows and performs a swap between these two elements if the first is larger than the second.

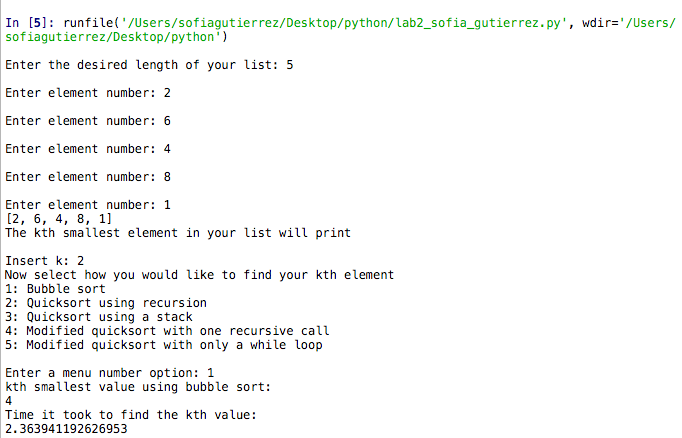
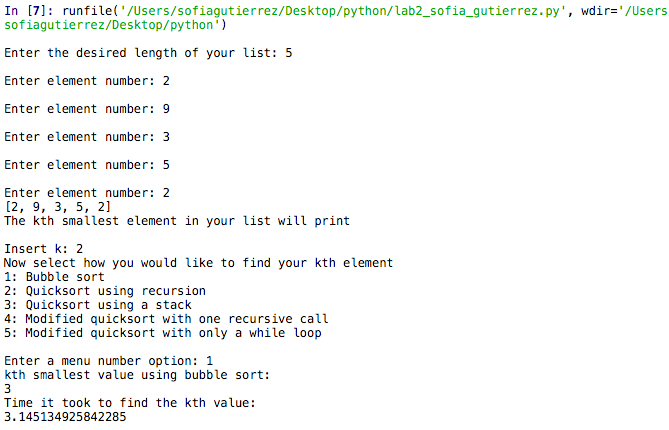
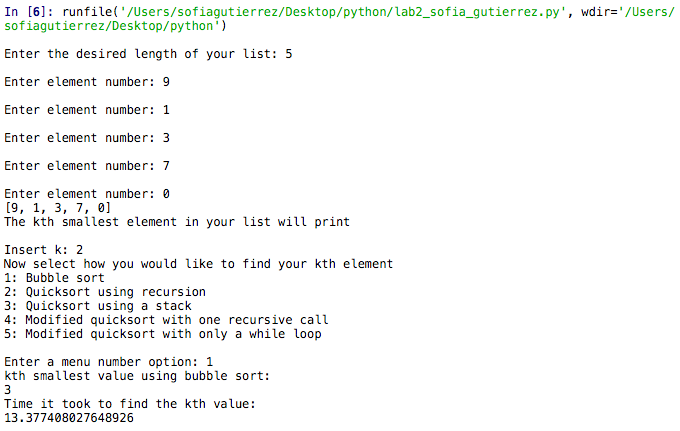
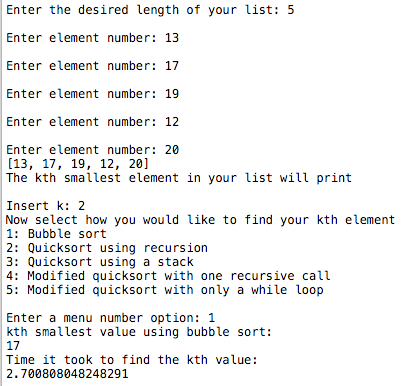
Next came Quicksort where two methods were necessary in order to sort the list given by the user. The method select\_quick contained an if statement that basically checked if the list included more than 1 element. Then partition, the second method necessary, was called carrying the list, a low variable, which stored 0 and a high variable that stored the length of the list, as its parameters. Within the partition method a new variable would be declared named ‘i’, which would store low-1, which was essentially -1 since low equaled 0. Then the pivot is created which is vital for quicksort. The pivot in these cases was the last element in the list. Next, a new variable ‘j’ would iterate through the length of the list. If the current element where ‘j’ was at was less than or equal to the pivot, ‘i’ would increment by 1 and the pivot would swap places with ‘j’. If ‘j’ was less than the pivot then ‘j’ would increment without swapping. When ‘j’ reached the end- not including the last element which was also the pivot- the pivot then switched its position so that it would be in front of ‘i’, or in other words i+1. The list would then be sorted where all the elements smaller than the pivot would be to the left of the it, and all the elements larger than the pivot would be to right of it. The new position of the pivot would then be returned back select\_quick where it would be recursively called within select\_quick. This position of pivot would then be sent back to further sort the list until the whole list was sorted.

The next method on the program also included Quicksort but instead of recursion, a stack would be implemented. The size would depend on the high or the length of the list minus the low which was 0 and would be added by 1. A variable stack would then be created which stored the element at 0 multiplied by the size variable. The top of stack was then initialized and the initial values of low and high were pushed into the stack. Next, it was necessary to pop or remove all the variables from stack until the top was less than 0. The method partition would then be called again this time passing the list, variables low and high as its parameters and again would perform its job to sort part of the list as done so previously. The position of pivot would again be returned and any of the elements found to the left of the pivot would be pushed to the stack. Then the elements that were to the right of the pivot would be pushed to the stack. The purpose of this was again to organize the list so the kth smallest element would be found. This method is essentially the same as select\_quick except that instead of recursion, a stack would be implemented.

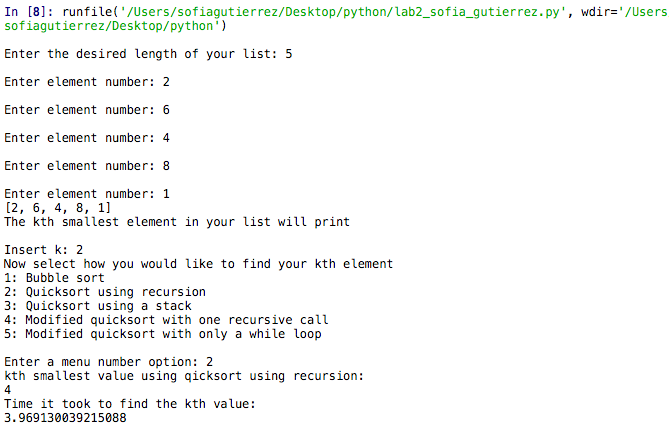
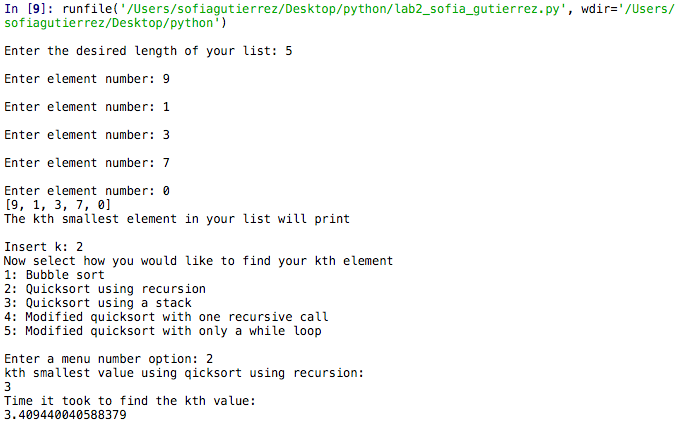
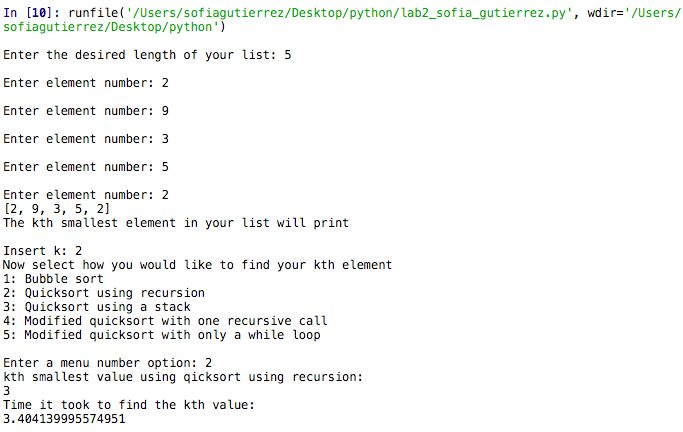
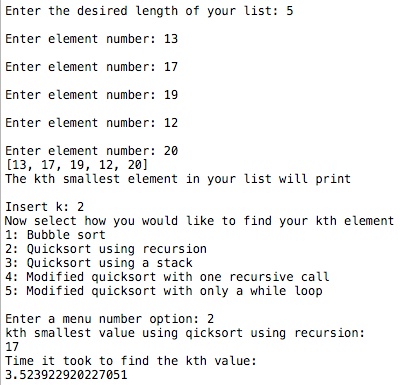
The next method to follow was the modified Quicksort which included one recursive call. This was basically the same as selcect\_quick only that instead of organizing both sides of the list, meaning the left partition and the right partition, the modified Quicksort method would only completely sort the side in which k was found. This would make it perform its task much quicker than select\_quick. The second modified Quicksort was included but instead of using one recursive call it would use no recursion and no stack. This method instead would only use a while loop. This was done by directly using the k variable. More specifically after the location of the pivot was returned by the partition method the method would immediately include a while loop with a condition that stated if the pivot position did not equal k. Inside this while loop would be an if statement that stated if k was less than the pivot position then the left side of the list would be sorted by partition and the pivot position would be updated. If the k variable was greater than the pivot then the right side of the list would be sorted by the partition method and again the pivot position would be updated.

After these methods were all included then came the print out of the menu options. The user would choose which sorting algorithm they most desired to find the kth smallest value in their list. After the user’s choice was inputted the results would print along with the time it took for the specific method to run.

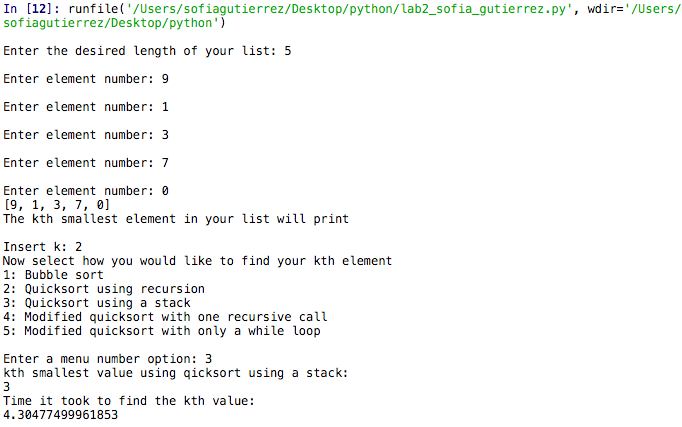
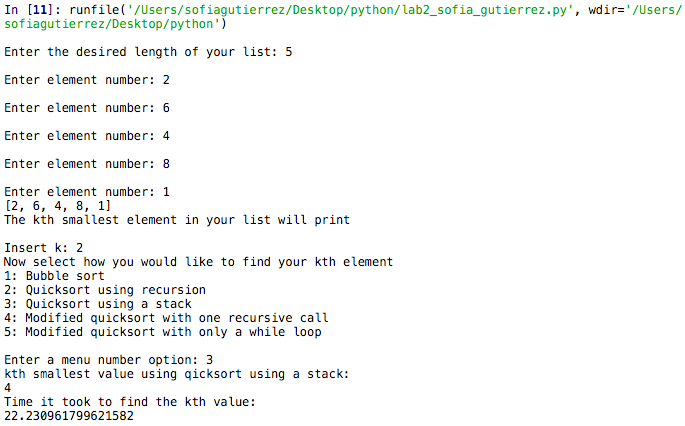
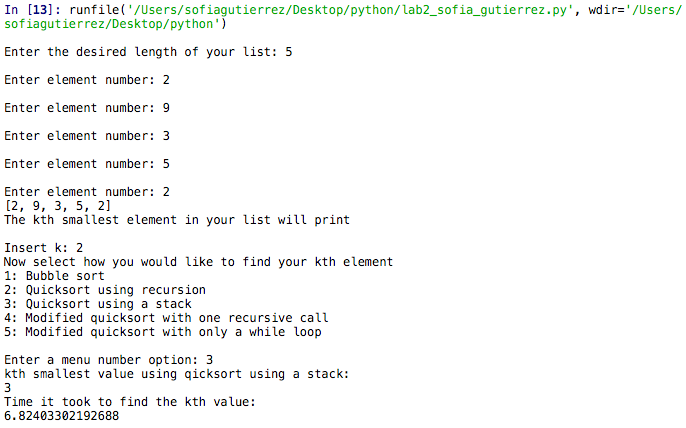
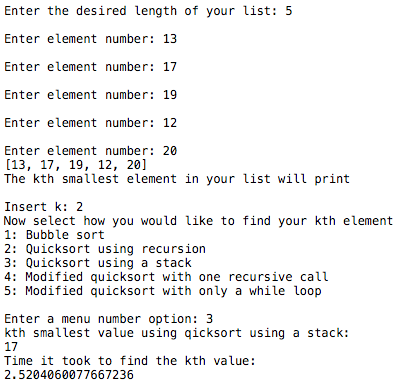
**Experimental Results**

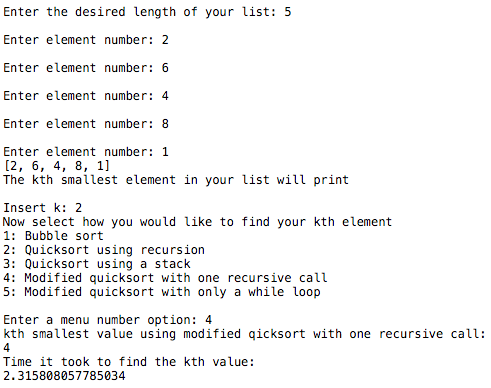
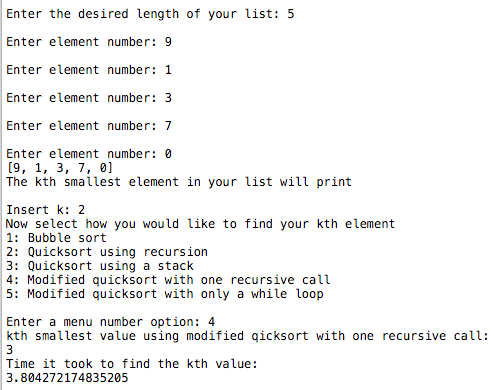
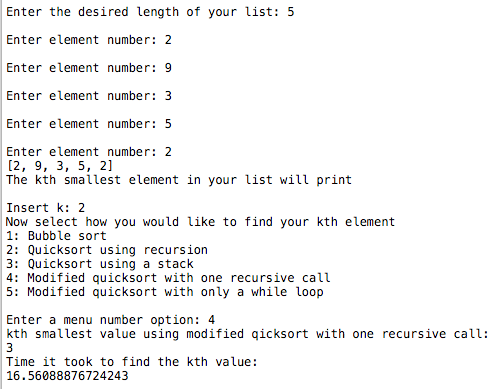
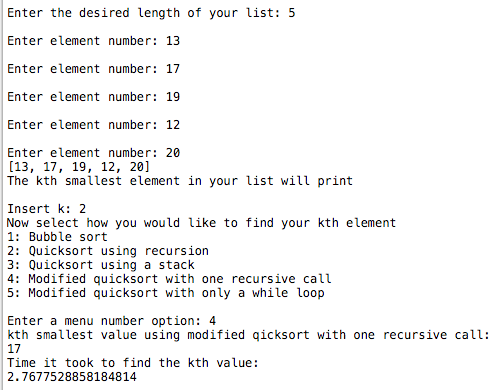
**The images display Bubble Sort outcomes using four different lists of the same length searching for the kth smallest integer.**

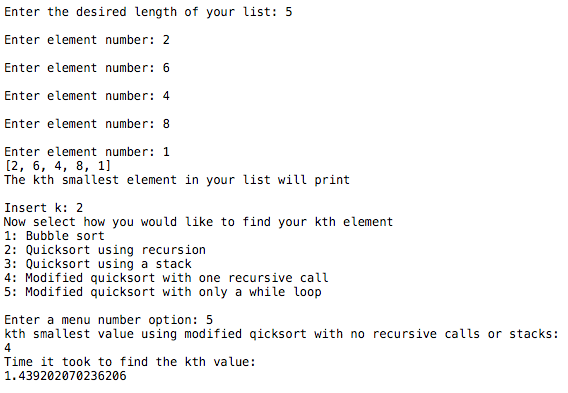
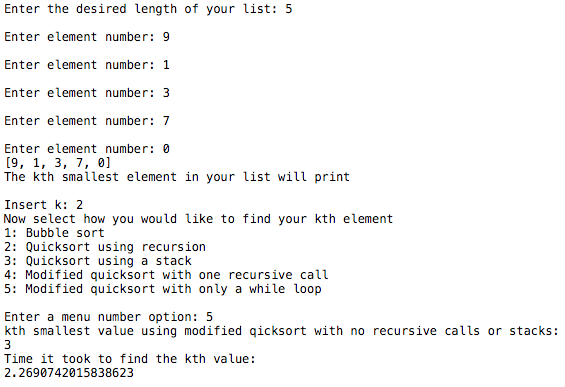
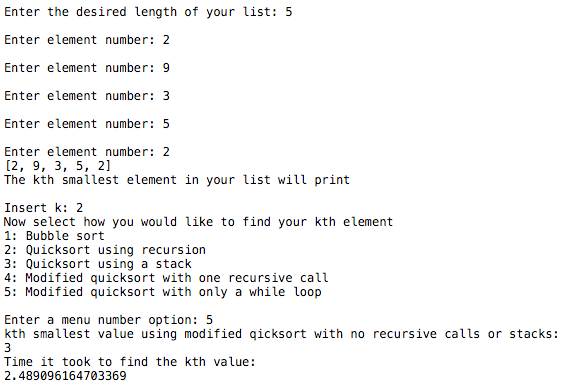
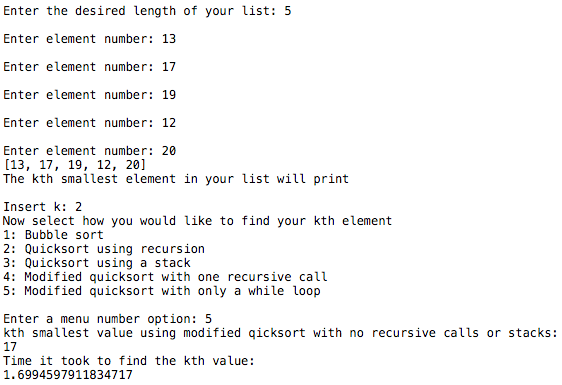
**The images display Quicksort (using recursion) outcomes using four different lists of the same length searching for the kth smallest integer.**

**The images display Quicksort (using a stack) outcomes using four different lists of the same length searching for the kth smallest integer.**

**The images display a modified version of Quicksort (with one recursive call) and its outcomes using three different lists of the same length searching for the kth smallest integer.**

**The images display a modified version of Quicksort (using while with no recursive call or stack) and its outcomes using three different lists of the same length searching for the kth smallest integer.**

**Bubble Sort**

The time it took for the kth value to be found in all lists that were inputted were somewhat similar except with the list that contained the elements [9, 1, 3, 7, 0] where the time increased by about 10 seconds.

**Quicksort (using recursion)**

The time it took for the kth value to be found with quicksort using recursion between all the lists given were very similar. In fact, they were all 3 seconds.

**Quicksort (using a stack)**

The time displayed for this quicksort varied greatly, ranging from 2 seconds all the way up to 22 seconds.

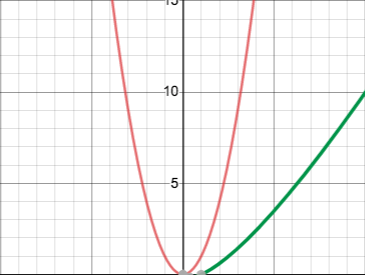
**Modified Quicksort (using one recursive call)**

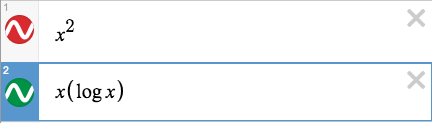
This method had times that were similar to that of Bubble Sorts in where 3 of the times for different lists were similar, averaging to a time of 2 seconds. The time for the list containing the elements [2, 9, 3, 5, 2] had a time of 16 seconds.

**Modified Quicksort (using while with no recursive call or stack)**

This method had similar times with all the lists that were given with times that ranged from 1 to 2 seconds.

Modified Quicksort (using while with no recursive call or stack) proved to be quicker than all the methods in the program which was expected considering it only ordered the half of the list where the kth smallest element was found. Also, instead of using recursion, it used a while loop, which is usually quicker than recursion. Furthermore, Bubble Sort should be slower than Quicksort. Bubble Sort has a time complexity of O(n^2) while Quicksort has a time complexity of O(n(log n)). Mathematically n^2 is greater than n(log n) and therefore Bubble Sort should be slower. This being said, the time displayed for Bubble Sort was not the slowest time in the program but instead, based on these test runs, the Quicksort method that used a stack was.



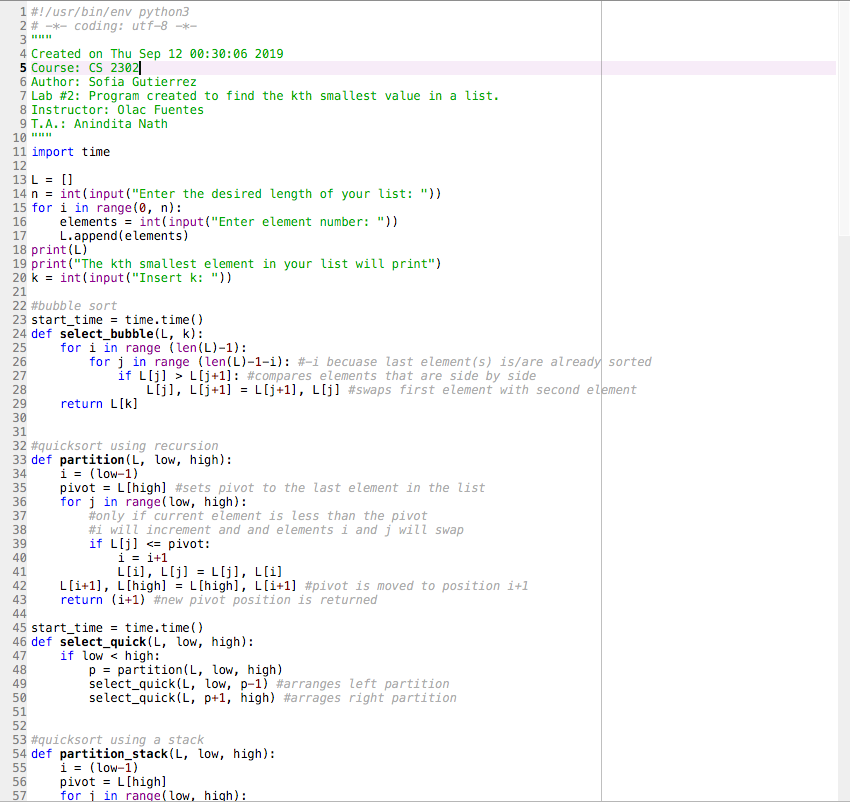


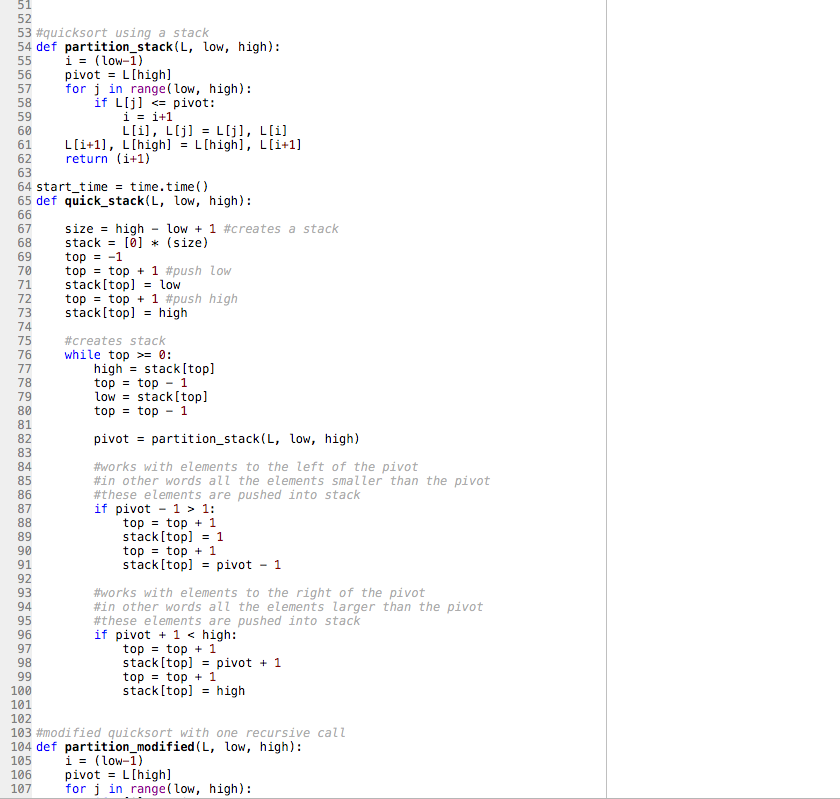
**The figure above displays n^2 represented by the red line and n(log n) represented by the green line. This shows n^2 is slower than n(log n).**

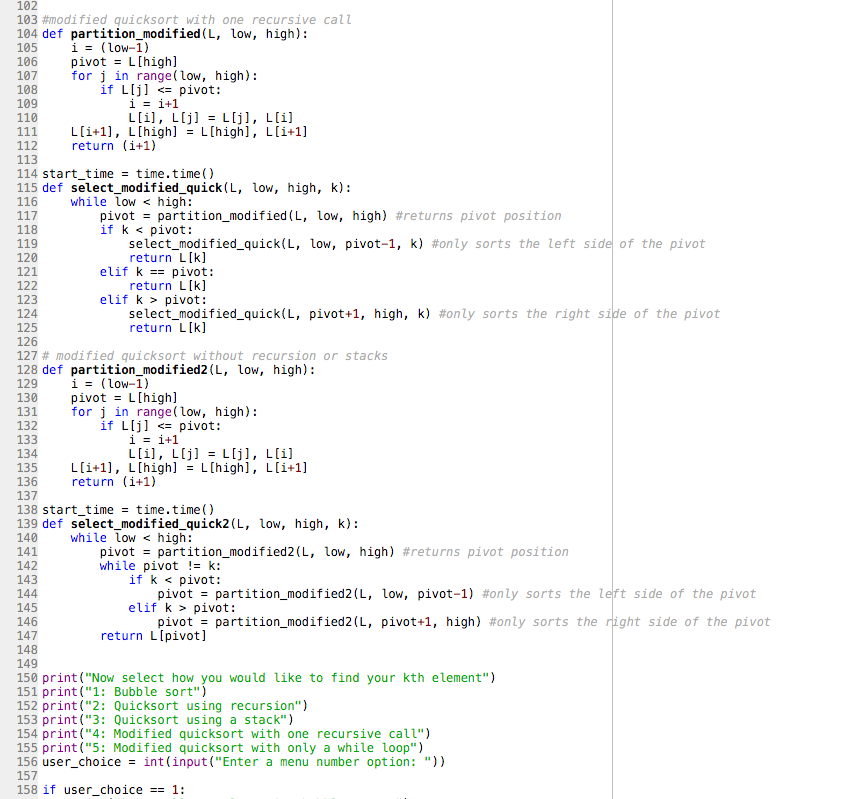
**Conclusion**

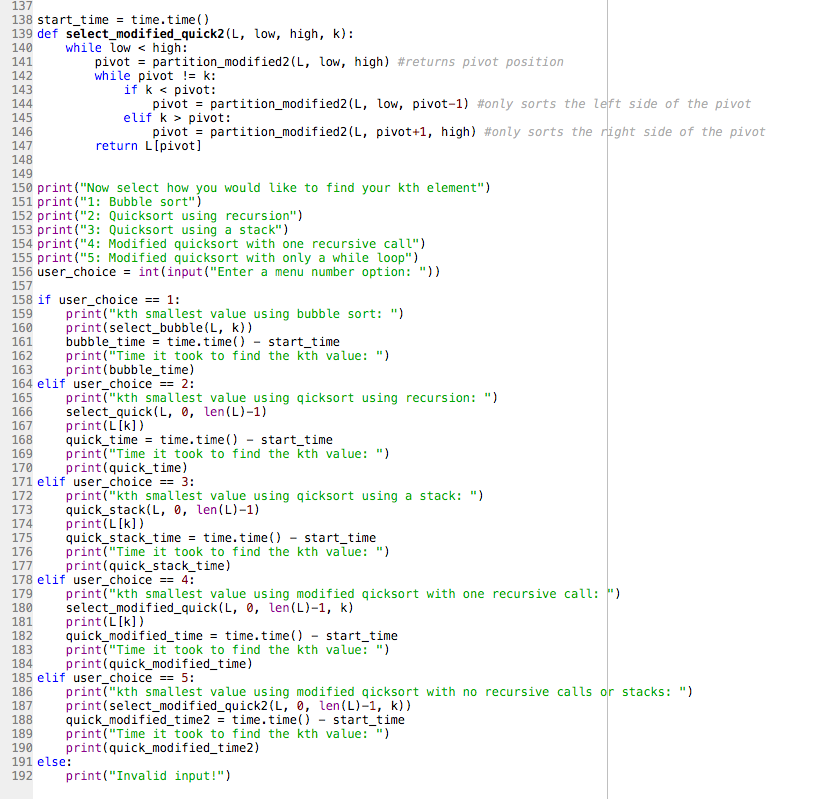
This program helped to clear the air in how two specific sorting algorithms work. The modified versions of Quicksort became useful in seeing how it operates and what is necessary to make it successful. There is not one right way to create a Quicksort method but instead there are a sort of guidelines. One of those guidelines includes to first identify the pivot, whether this be the first, median, or last element, as long as the position of the pivot is consistent then the list should be sorted successfully. This also helped in identifying the time complexities of the of Bubble Sort and Quicksort and how there is a big difference in time if the wrong sorting algorithm is implemented.

**Appendix**









**Academic Honesty Certification**

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.

Sofia Gutierrez September 24, 2019