**CS 2302 Data Structures**

**Fall 2019**

**Lab Report #2**

Due: September 20th, 2019

Professor: Olac Fuentes

TA: Anindita Nath

Author: Kevin Ramirez

**Introduction**

For our second lab assignment, our objective was to make a multiple different implementation of both bubble sort and quick sort, and each method must return the smallest “kth” element, “k” being a user integer input. With the exception the for the bubble sort it was only ask of us to do it normally,” select\_bubble” with no modifications. For the quick sort, we had to make multiple different implementations of it, one normally implemented with two recursive calls,” select\_quick”, the next a modified version of the previous quick sort, “select\_modified\_quick”, a normal quick sort algorithm only that instead of using two recursive calls it only uses one. For the second part of the lab, there are two methods for quick sort. The first one requires the use of stacks, instead of recursion, the next requires while loops, and no recursion or stacks.

**Proposed Solution Design and Implementation**

**Operation #1:**

The first operation, which was just a simple bubble sort, I decided to go for the basic implementation of it, and making two method which were related to each other one was the sorting algorithm and the other was the broadcaster, calling the method and returning whatever element k is. The bubble sorting algorithm works as following; first it checks if the first element on the list is smaller or grater than the second element, if yes, it moves the pointer to index [1], and index [2], then it checks those two. If the first element is bigger than the second one it is sent to the back of the list. And this process is repeated by the length of the list squared. The big O of bubble sort is n^2. O = N^2

**Operation #2:**

The second operation of part 1 had to do with implementing a normal quick sort algorithm, and as such I decide to just add the normal quick sort with two recursive calls, for this one I created three methods, “Quick\_parti” the partition method of quick sort, which primarily focuses in dividing the list in half then arrange some parts of the list, and returns the next pivot number. The second method is, “quick\_sort” this one takes care of making the first partition of the list and two recursive calls which create other partitions and put the list back together now sorted. Finally, the broadcaster “Q\_sort\_call”, this method just handles the call for quick\_sort and returns the appropriate ‘k’ value from the list. The big O notation for this one is O = n(log(n))

**Operation #3:**

The third operation of the first part is the modified quick sort algorithm which only uses one recursive call instead of two. The way that I tried to make this work was by only activation either of the calls for the partition method, depending on whether if the K was bigger than the current pivot of the first partition, meaning that if the firs pivot that we got from the starting partition was smaller then the k element that we wish for we need to continue to arrange the list until both k and the pivot are the same value. Once we know that k and the pivot are the same, we know that we can stop and return the sorted list plus the element at k given by the user. The big O notation of this is O = N^2.

**Operation #4:**

For the quick sort that utilizes only stacks, I was unable to complete it. My method would return the correct k element but it would not correctly sort the list.

**Operation #5:**

Next, for the fourth operation, it is the implementation of quick sort by only using while loops and no recursion. The only way I could make this work was by creating a mid-point with reference to the middle of the list minus 1, the logic is pretty much the same as for operation#3, so the operation will continue until the pivot that was chosen equals the mid-point. The big O notation for this one is O = n(log(n))

**Experimental Results**

The test cases for these operations are, an empty list, a list with only one element, and a list with 10 elements. And for the user k inputs, are 0, 3 , and len(list)-1.

**Operation #1:** **Bubble Sort**

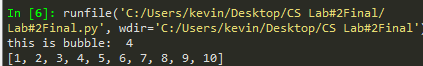
Case #1: Empty list: [ ], K=0

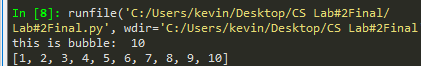


Case #2: One element on list: [1], K=0



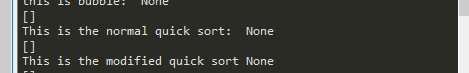
Case #3: Ten elements on list: [10,9,8,7,6,5,4,3,2,1], K=3



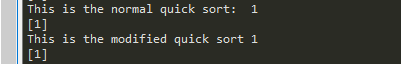
Case #4: Ten elements on list: [10,9,8,7,6,5,4,3,2,1], K=len(list)-1

**Operation #2: Quick Sort**

Case #1: Empty sort: [0], K = 0



Case #2: One element on list: [1], K=0



Case #3: Ten elements on list: [10,9,8,7,6,5,4,3,2,1], K=3



Case #4: Ten elements on list: [10,9,8,7,6,5,4,3,2,1], K=len(list)-1



**Operation #3: Modified Quick Sort**

Case #1: Empty sort: [0], K = 0



Case #2: One element on list: [1], K=0



Case #3: Ten elements on list: [10,9,8,7,6,5,4,3,2,1], K=3

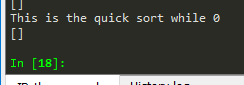


Case #4: Ten elements on list: [10,9,8,7,6,5,4,3,2,1], K=len(list)-1



**Operation #5: Quick Sort While:**

Case #1: Empty sort: [0], K = 0



Case #2: One element on list: [1], K=0



Case #3: Ten elements on list: [10,9,8,7,6,5,4,3,2,1], K=3



Case #4: Ten elements on list: [10,9,8,7,6,5,4,3,2,1], K=len(list)-1

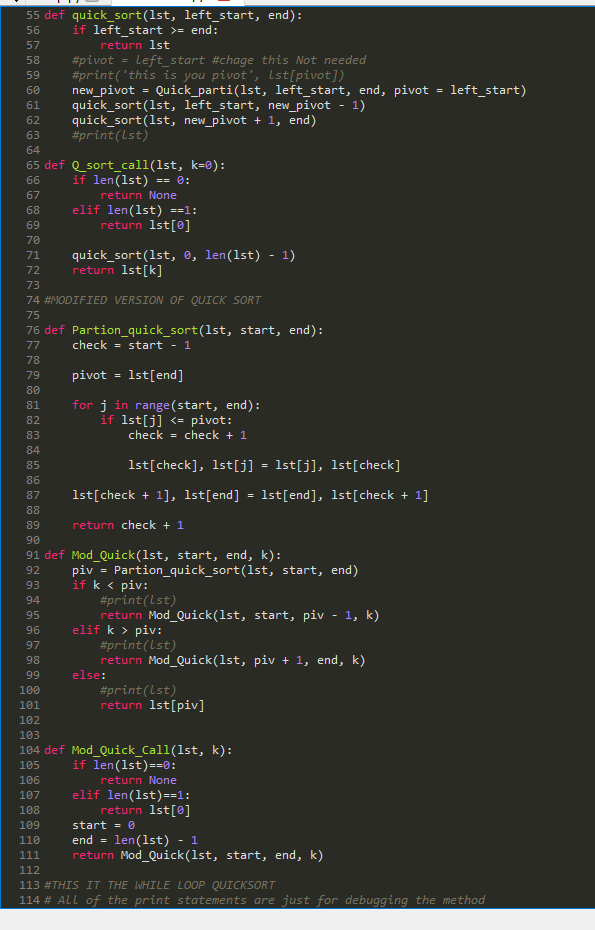


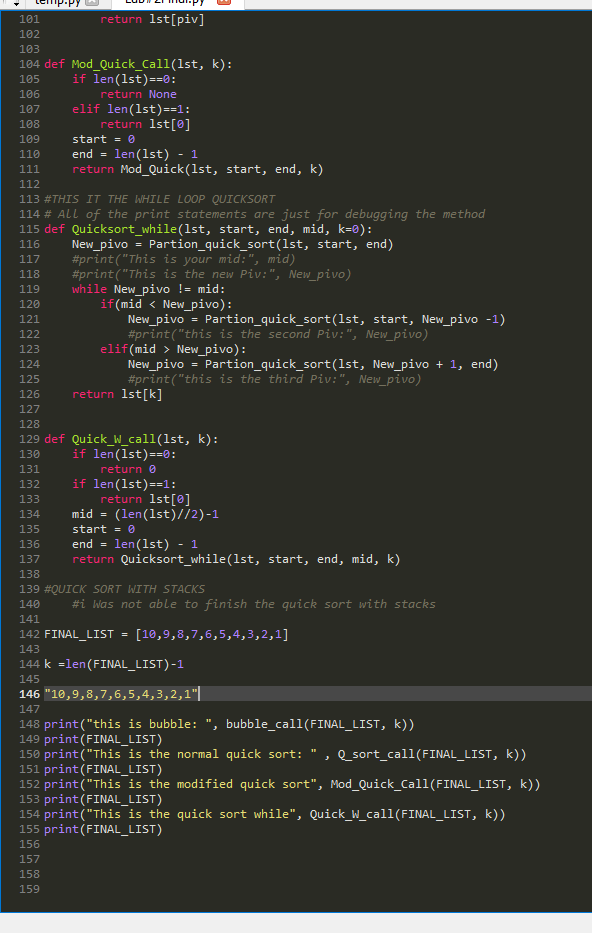
**Conclusion**

With this lab, I learned how to implement quick sort in multiple different ways, despite the fact that I was not able to finish the quick method that uses stacks. It was interesting to see how many ways you can implement one quick sort in all different ways. Regarding the code for quick sort with stacks I was able to finish something but given the fact that it does not work properly I decide to just leave it out for this lab.

**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