CS 2302

Lab 1 Report

Author: David Ayala

ID: 80617838

Instructor: Olac Fuentes

TA: Anindita Nath

**Introduction**

The purpose of this lab was to the following algorithms (bubble sort, quick sort, merge sort and modified quick sort) to find the median of a List. This list was created by randomly generating numbers and appending them to the List. All of the algorithms should return the same median number other wise there is something wrong.

**Proposed Solution & Design Implementation**

**Bubble Sort**

Honestly this bubble sort is the one that Professor Fuentes gave us, and I couldn’t figure out how to condense it anymore so after understanding how it worked, it helped me be able to do the next three methods. And I used his as a base case to see if my other methods were working correctly.

**Quick Sort**

I first acknowledged that a pivot point was necessary then proceeded to split the list into that way it could sort according to the value of the pivot point and the current values the item that is being compared to it. This is how it sorts the list into different categories such is less than pivot or is greater than pivot. This was possible to professor Fuentes talking about the basic idea of quick sort in class which help me come up with how to implement.

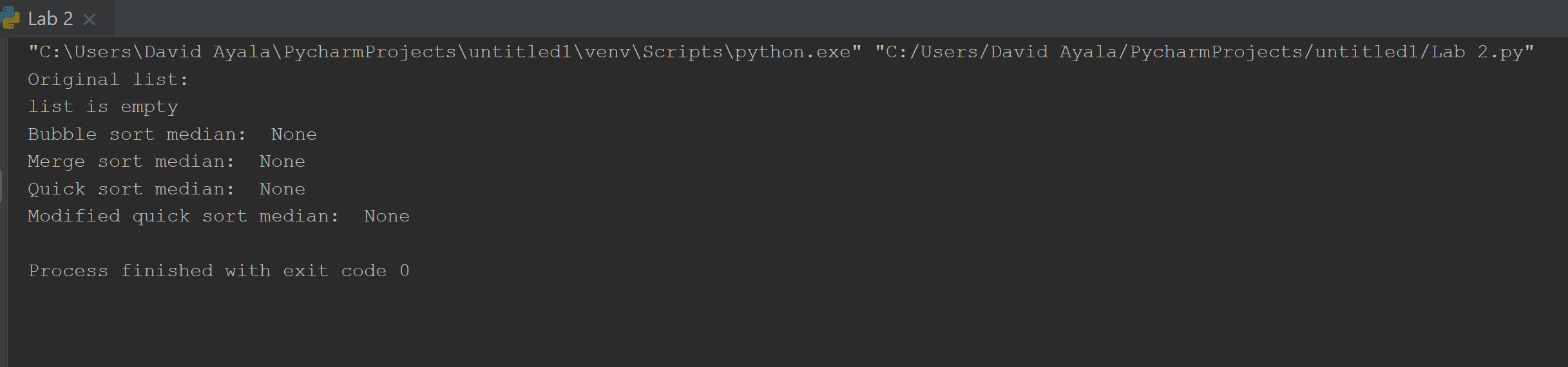
**Merge Sort**

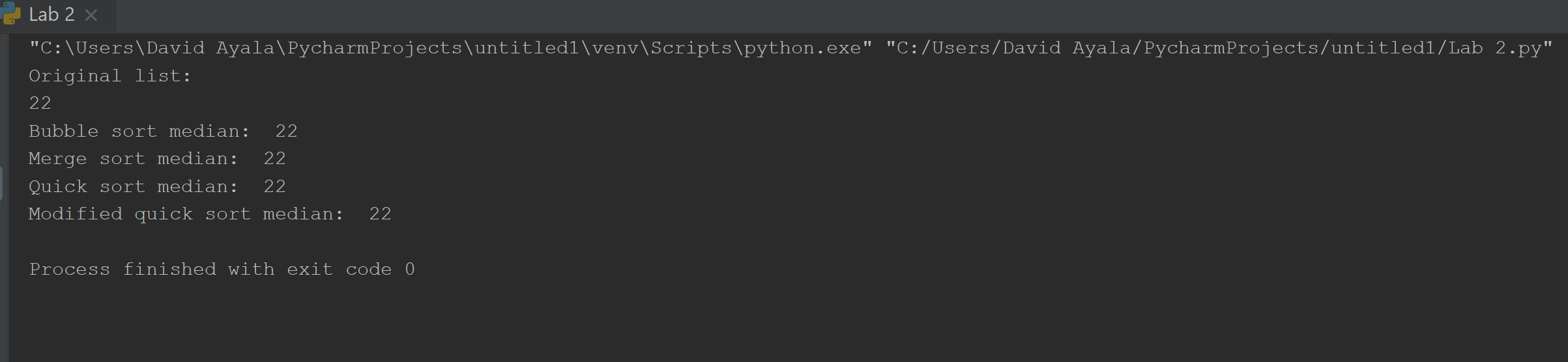
I understood that the Merge function was similar to the quick sort function so I applied a similar logic to the merge sort except this list would be split up, then compared to the values of each other and then merged together back to the original is list but sorted. I accomplished this with two methods MergeSort method and the Merge method the first one is to split the list into two list and the second one is to compare the values of each list VS each other and place them back into the original list that is now empty thanks to the MergeSort method that makes it empty.

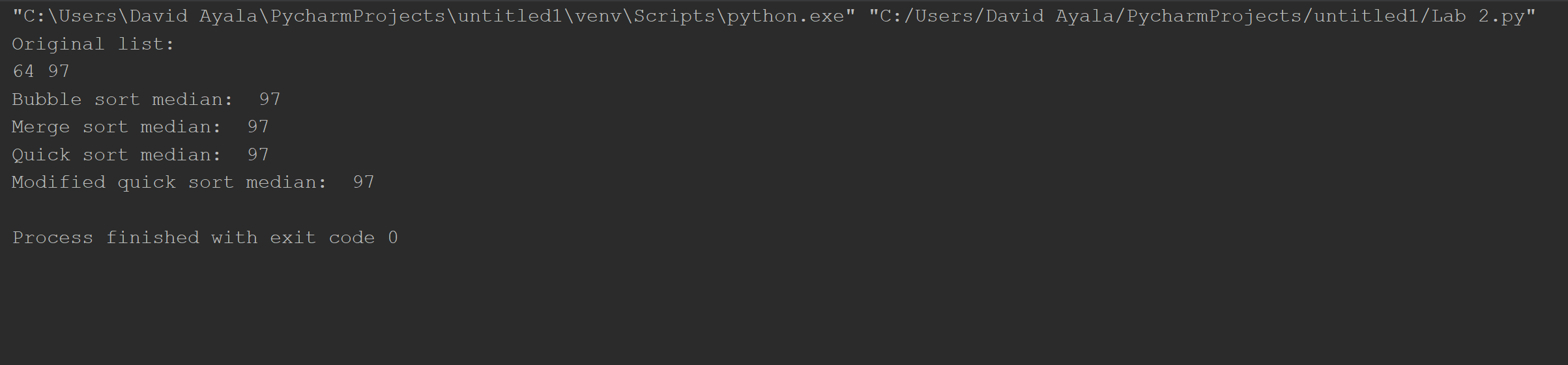
**Modified Quick Sort**

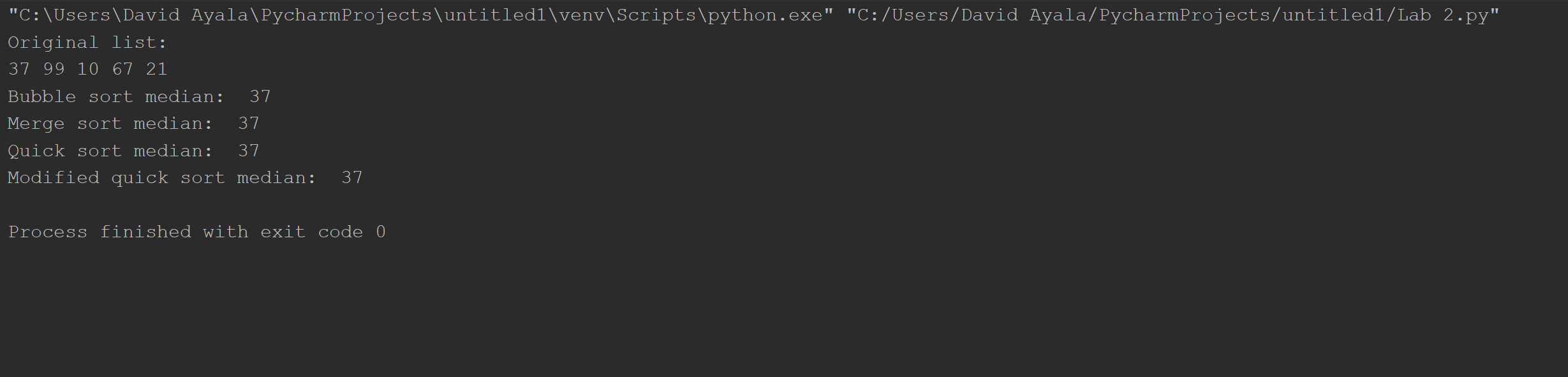
The modified Quick sort is almost the same as the regular quick sort except that after it splits the list into two then it searches through the first list and if the estimated mid-point is located in the first half of the list then it returns the first half and the midpoint is there. But if the midpoint is not in the first half of the list then it goes to the second half of the list because the midpoint has to be there. Sorry if I did not explain this too well, Professor Fuentes explains it much better and that what helped me program it.

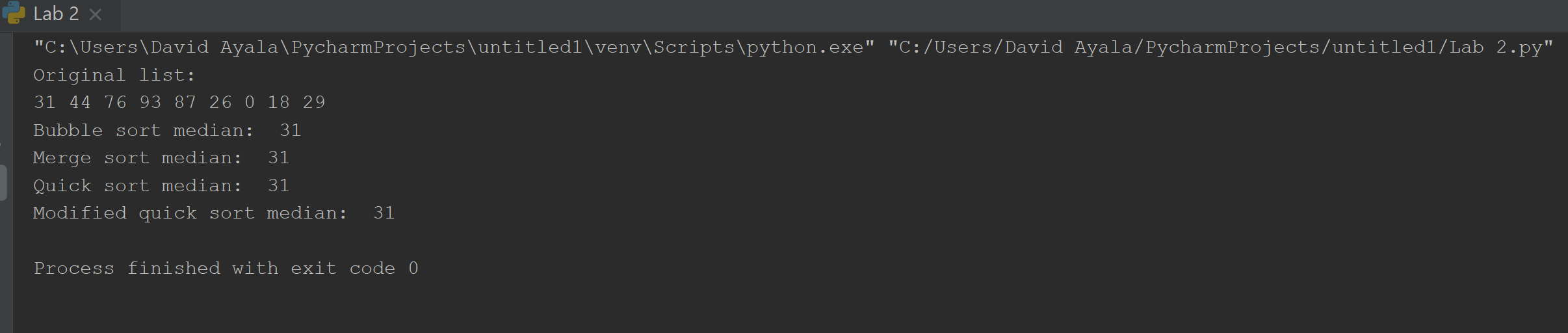
**Experimental Results**

****

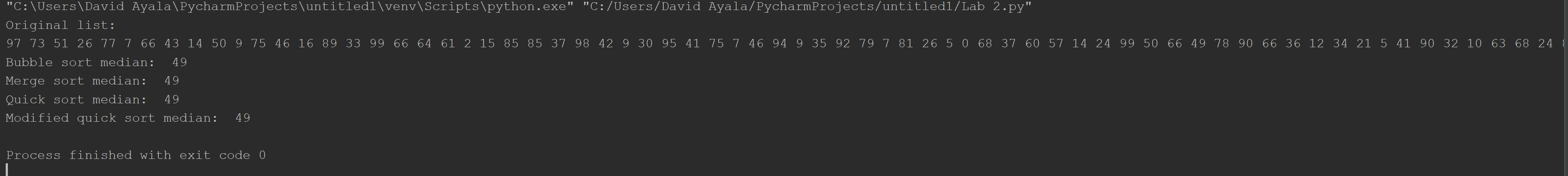
****

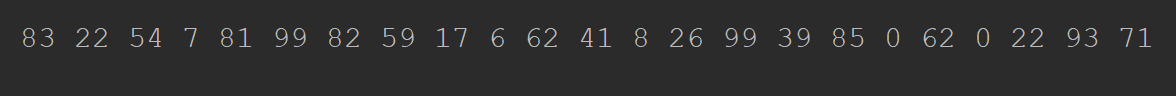
****





This next one was like 90+ numbers so it was too long for one picture.





**Conclusion**

This was a very hard project to due to the fact that we basically had to create the methods from scratch, luckily Fuentes gave us Bubble Sort so that help a lot because otherwise I don’t think I would have been able to finish this lab. I can see why it is important for us to learn this because we never know when we may need to use these methods again. Also bubble sort is horrible for the larger that the list gets, the others depend because they are (very) close to each as the list gets bigger.

**Appendix**

import random

class Node(object):

def \_\_init\_\_(self, item, next=None):

self.item = item

self.next = next

class List(object):

# Constructor

def \_\_init\_\_(self):

self.head = None

self.tail = None

self.Length = 0

def IsEmpty(L):

return L.head == None

def Append(L, x):

L.Length += 1

if IsEmpty(L):

L.head = Node(x)

L.tail = L.head

else:

L.tail.next = Node(x)

L.tail = L.tail.next

def Prepend(L, x):

L.Length += 1

if IsEmpty(L):

L.head = Node(x)

L.tail = L.head

else:

L.head = Node(x, L.head)

def Print(L):

if IsEmpty(L):

return print('list is empty')

else:

temp = L.head

while temp is not None:

print(temp.item, end=' ')

temp = temp.next

print()

def IsSorted(L):

if IsEmpty(L):

return None

temp = L.head

while temp.next is not None:

if temp.item > temp.next.item:

return False

temp = temp.next

return True

def BubbleSort(L):

if IsEmpty(L):

return None

elif IsSorted(L):

return L

else:

change = True

while change:

change = False

temp = L.head

while temp.next is not None:

if temp.item > temp.next.item:

temp2 = temp.next.item

temp.next.item = temp.item

temp.item = temp2

change = True

temp = temp.next

def QuickSort(L):

if IsEmpty(L):

return None

elif IsSorted(L):

return L

else:

pivot = L.head.item

list1 = List()

list2 = List()

temp = L.head.next

while temp is not None:

if temp.item < pivot:

Append(list1, temp.item)

else:

Append(list2, temp.item)

temp = temp.next

QuickSort(list1)

QuickSort(list2)

Prepend(list2, pivot)

if IsEmpty(list1):

L.head = list2.head

L.tail = list2.tail

else:

list1.tail.next = list2.head

L.head = list1.head

L.tail = list2.tail

def MergeSort(L):

if IsEmpty(L):

return None

elif IsSorted(L):

return L

else:

list1 = List()

list2 = List()

temp = L.head

for i in range(L.Length // 2):

Append(list1, temp.item)

temp = temp.next

while temp is not None:

Append(list2, temp.item)

temp = temp.next

MergeSort(list1)

MergeSort(list2)

L.head = None

L.tail = None

L.Length = 0

merge(L, list1, list2)

def merge(L, list1, list2):

temp1 = list1.head

temp2 = list2.head

while temp1 and temp2 is not None:

if temp1.item < temp2.item:

Append(L, temp1.item)

temp1 = temp1.next

else:

Append(L, temp2.item)

temp2 = temp2.next

if temp2 is None:

while temp1 is not None:

Append(L, temp1.item)

temp1 = temp1.next

if temp1 is None:

while temp2 is not None:

Append(L, temp2.item)

temp2 = temp2.next

def ModifiedQuickSort(L, middlePosition):

if IsEmpty(L):

return None

elif L.Length == 1:

return L.head.item

else:

pivot = L.head.item

list1 = List()

list2 = List()

temp = L.head.next

while temp is not None:

if temp.item < pivot:

Append(list1, temp.item)

else:

Append(list2, temp.item)

temp = temp.next

if list1.Length > middlePosition:

return ModifiedQuickSort(list1, middlePosition)

elif (list1.Length == middlePosition):

return pivot

else:

return ModifiedQuickSort(list2, middlePosition - list1.Length-1 )

def Copy(L):

copy = List()

temp = L.head

while temp is not None:

Append(copy, temp.item)

temp = temp.next

return copy

def BubbleSortMedian(L):

Clone = Copy(L)

BubbleSort(Clone)

temp = Clone.head

if IsEmpty(Clone):

return

else:

for i in range(Clone.Length // 2):

temp = temp.next

return temp.item

def MergeSortMedian(L):

Clone = Copy(L)

MergeSort(Clone)

temp = Clone.head

if IsEmpty(Clone):

return

else:

for i in range(Clone.Length // 2):

temp = temp.next

return temp.item

def QuickSortMedian(L):

Clone = Copy(L)

QuickSort(Clone)

temp = Clone.head

if IsEmpty(Clone):

return

else:

for i in range(Clone.Length // 2):

temp = temp.next

return temp.item

def ModifiedQuickSortMedian(L):

Clone = Copy(L)

if IsEmpty(Clone):

return

else:

return (ModifiedQuickSort(Clone, Clone.Length // 2))

randomGeneratedList = List()

for i in range(random.randrange(11)):

Prepend(randomGeneratedList, random.randrange(100))

print('list: ')

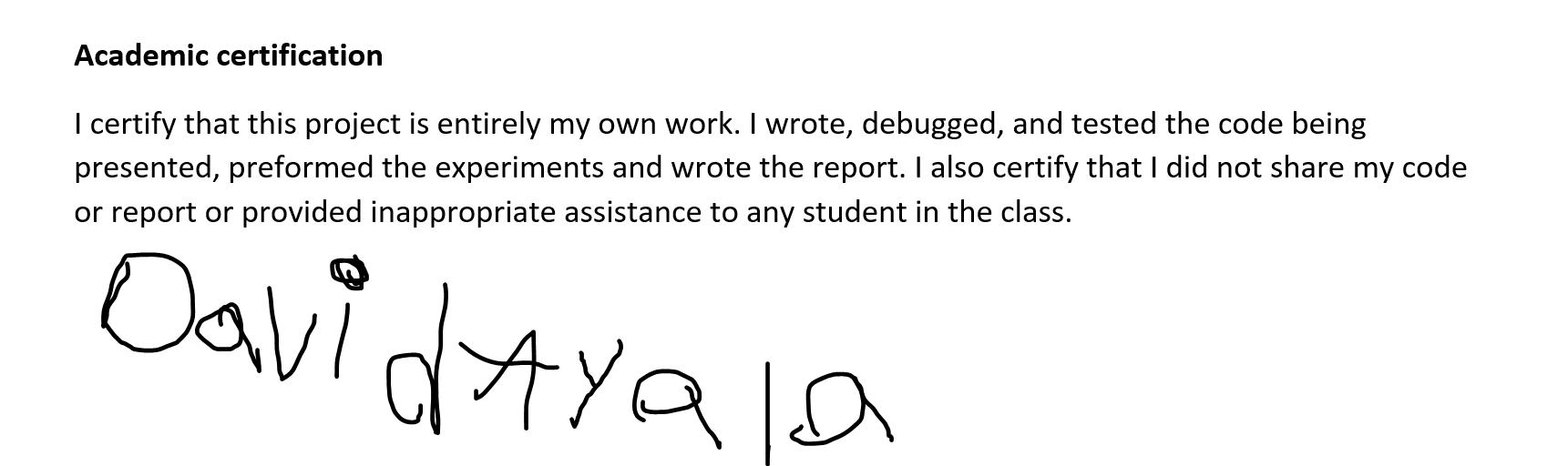
Print(randomGeneratedList)

print('Bubble sort median: ', BubbleSortMedian(randomGeneratedList))

print('Merge sort median: ', MergeSortMedian(randomGeneratedList))

print('Quick sort median: ', QuickSortMedian(randomGeneratedList))

print('Modified quick sort median: ', ModifiedQuickSortMedian(randomGeneratedList))



David Ayala

2/27/19