Samuel Chong

2/28/19

CS 2302

Lab 2

MW 10:30-11:50

Olac Fuentes

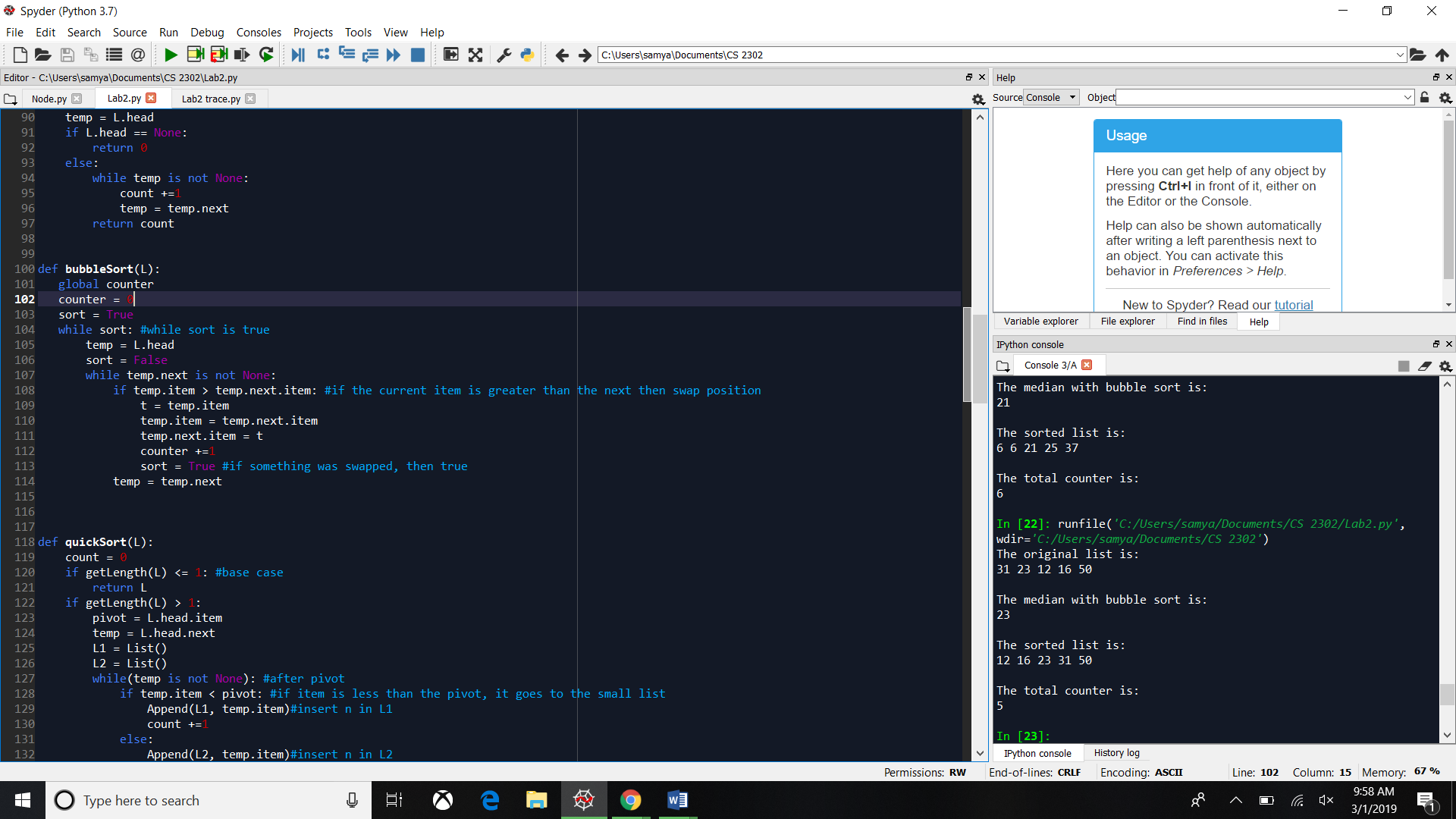
Introduction:

The problem we needed to solve for this lab was to sort a list using different types of sorting methods. We used bubble sort, quick sort and merge sort.

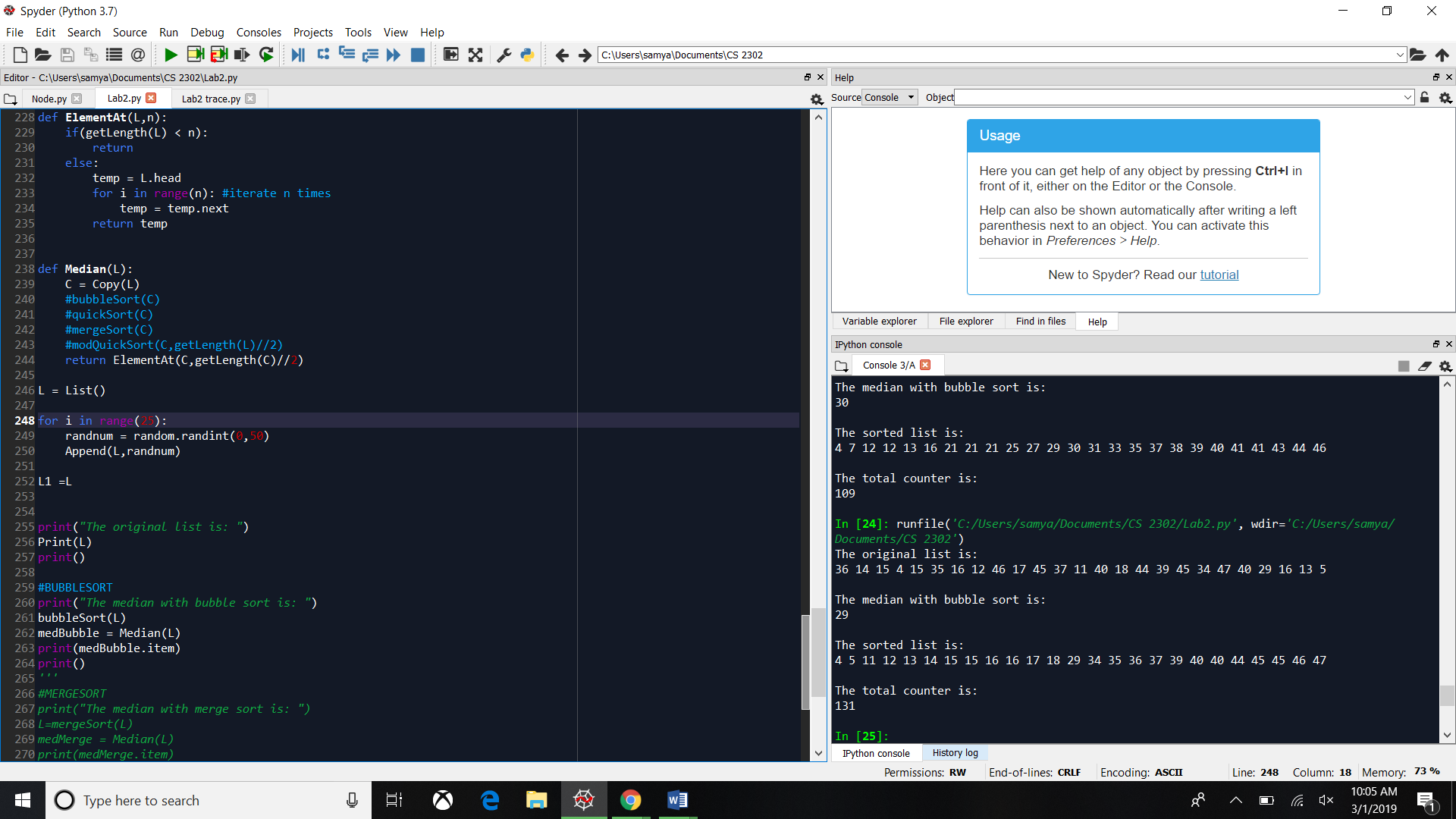
Bubble sort:

The way that bubble sort works is it traverses in order and if the first item is bigger than the second then they switch and so on until every element is in order. Luckily for us, the professor gave us the pseudo code for every sorting method and we just needed to translate it to code. The big O for bubble sort is O(n^2). This means this is the slowest sorting method compared to the others.

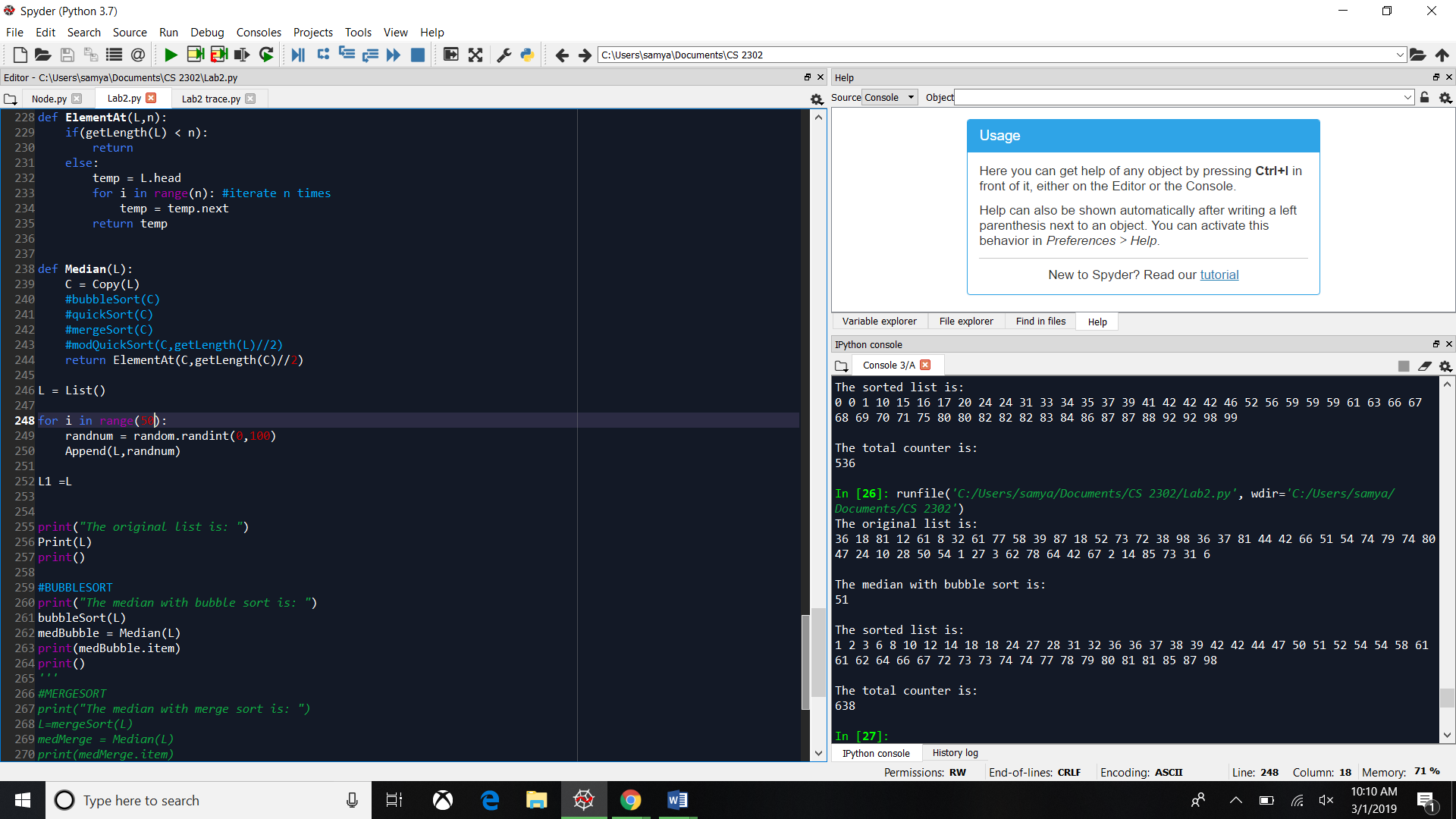
List size: 5



List size: 25



List size: 50

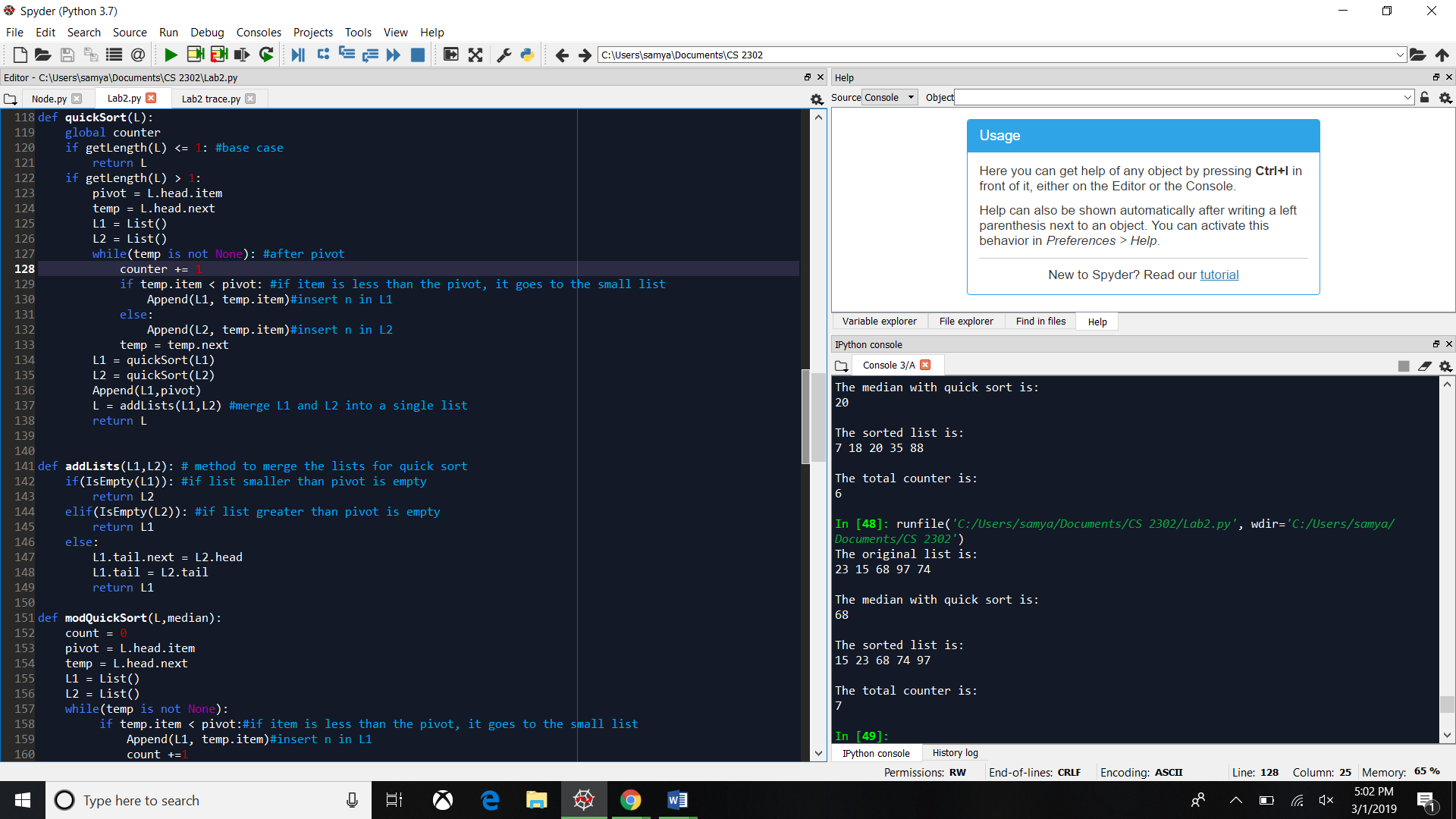


srot

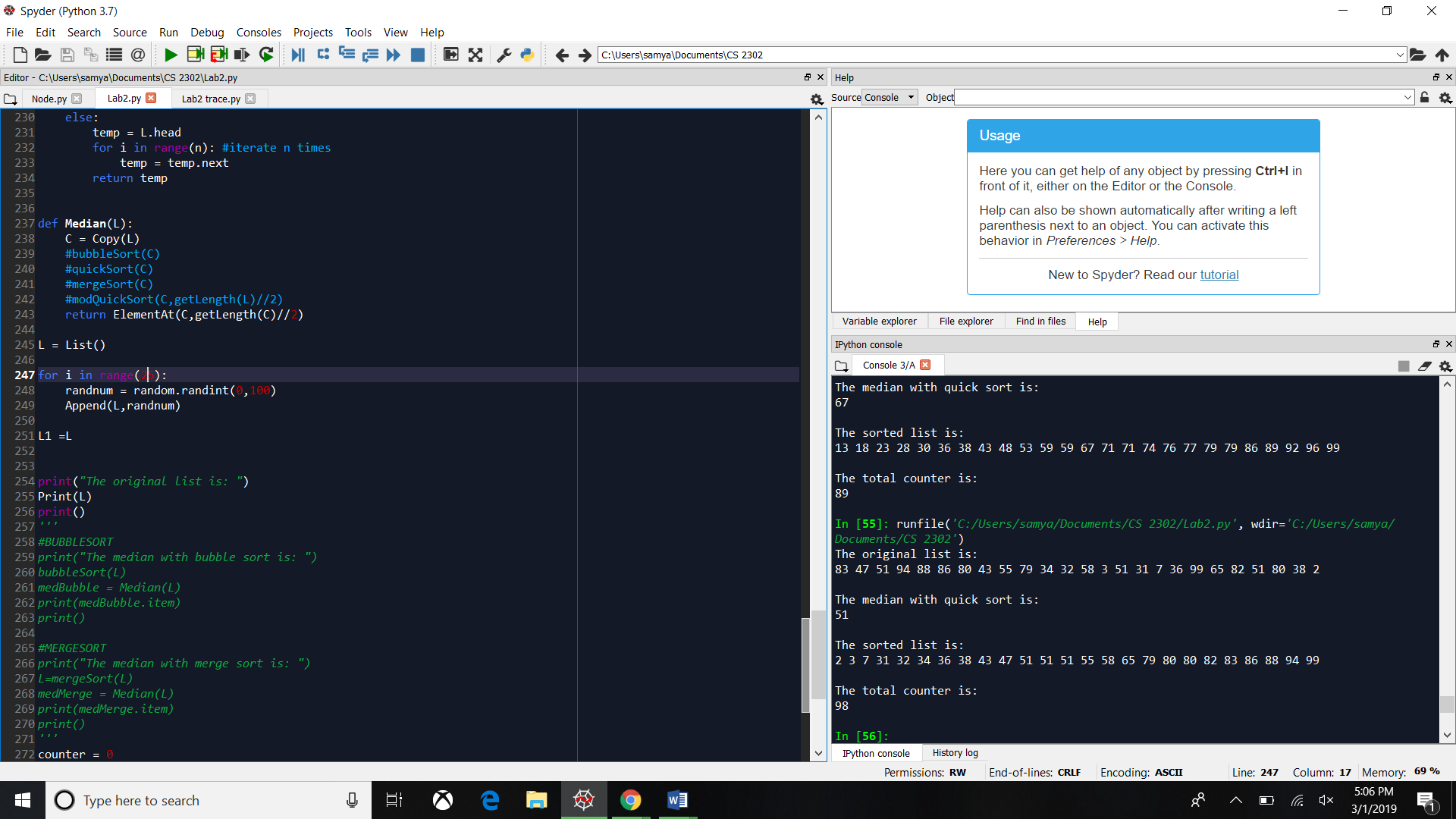
Quick Sort:

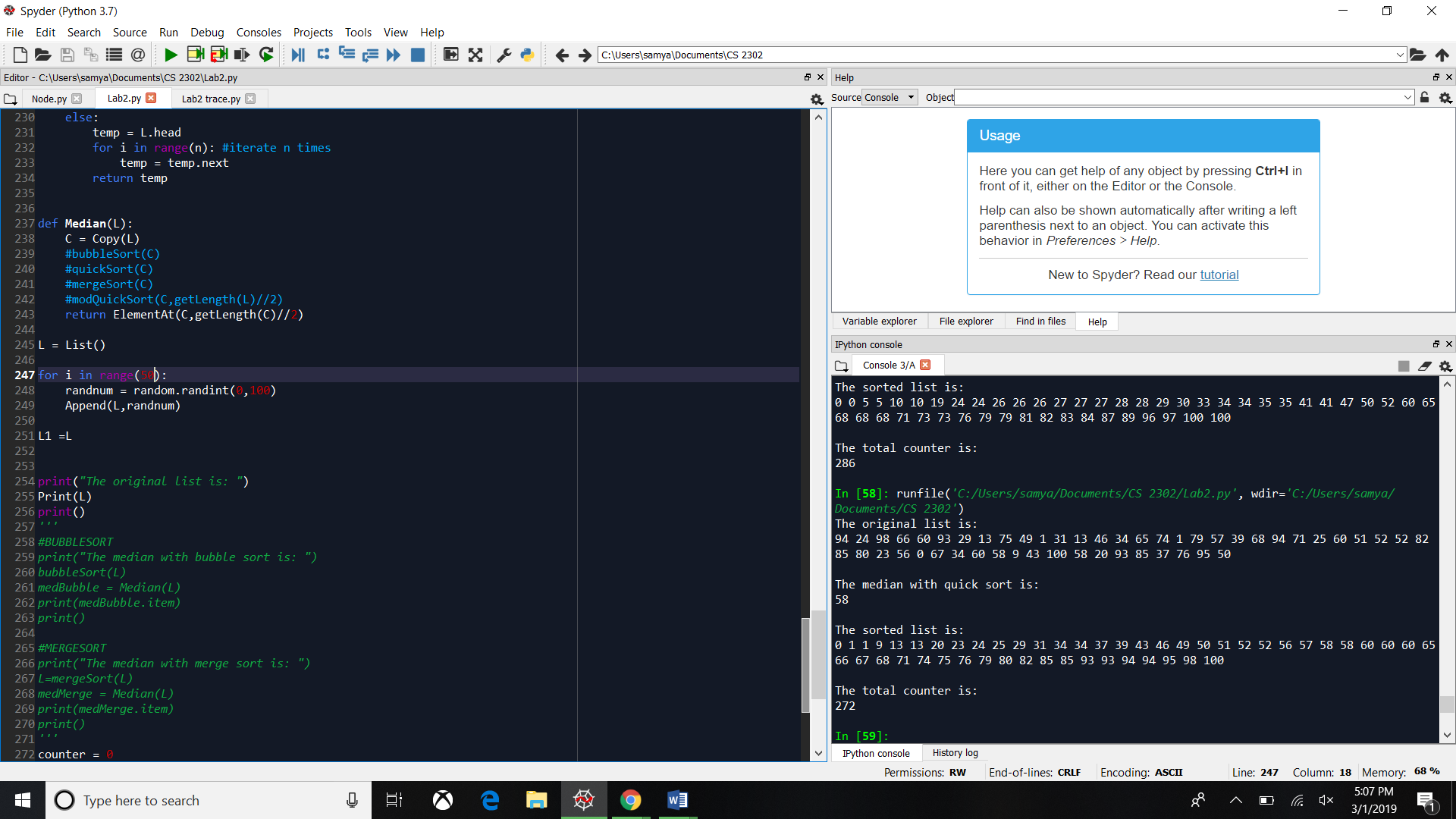
Quick sort is faster than bubble sort because it divides the original list into 2. The way they are divided is you need a pivot in my case is the lists head, and if the temp is smaller then it goes to L1, which is the list where the smaller items go to, and if the item is larger than the pivot, it goes to L2 which is the list for bigger numbers. Since we are using recursion, the recursion is going to keep making sub lists until the length of the lists are less or equal to 1. Once that has finished, we append the pivot to L1 so we don’t lose the link with the original list. The big O for quick sort is O(nlogn).

List size: 5



List size: 25

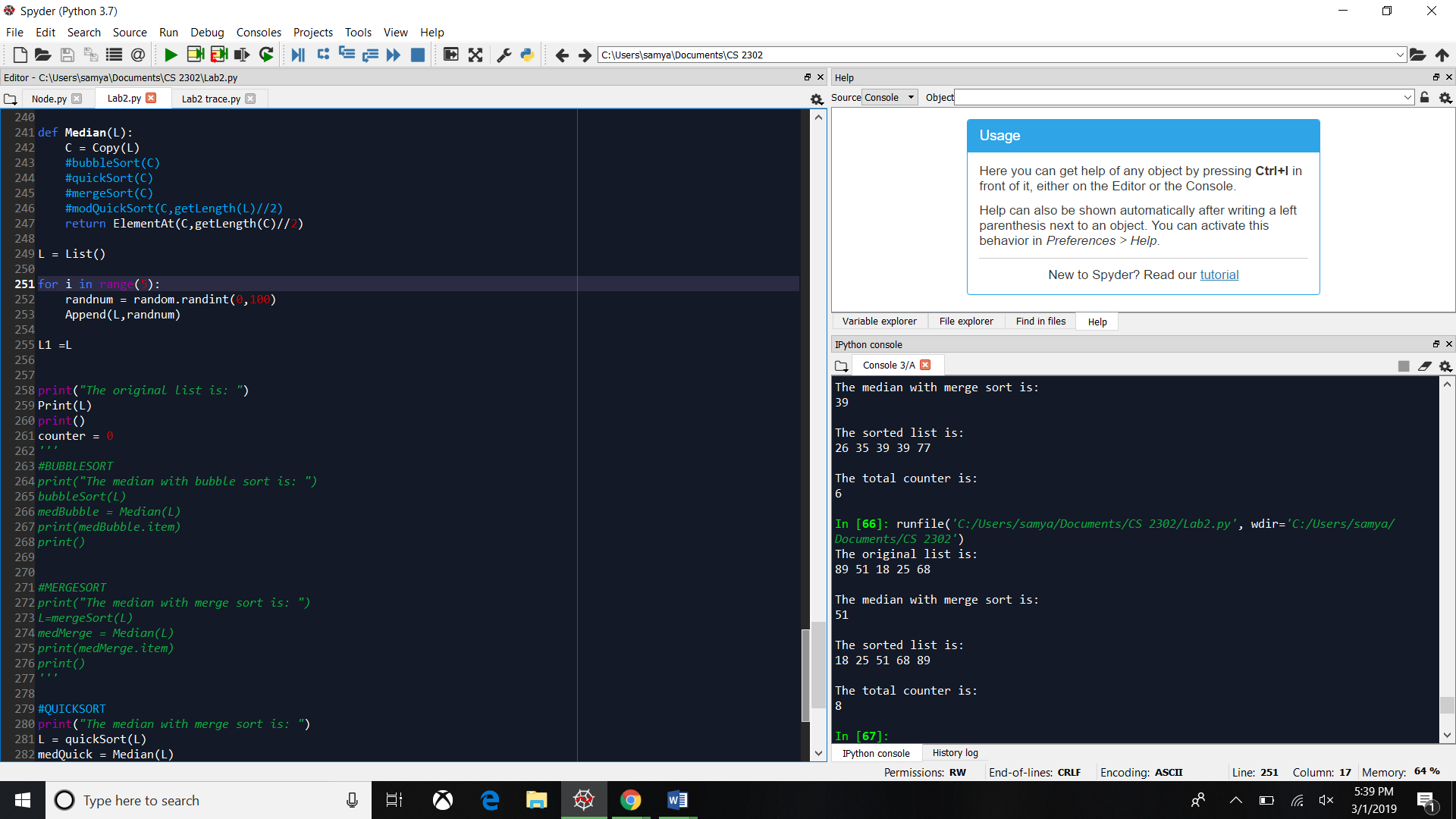


List size: 50

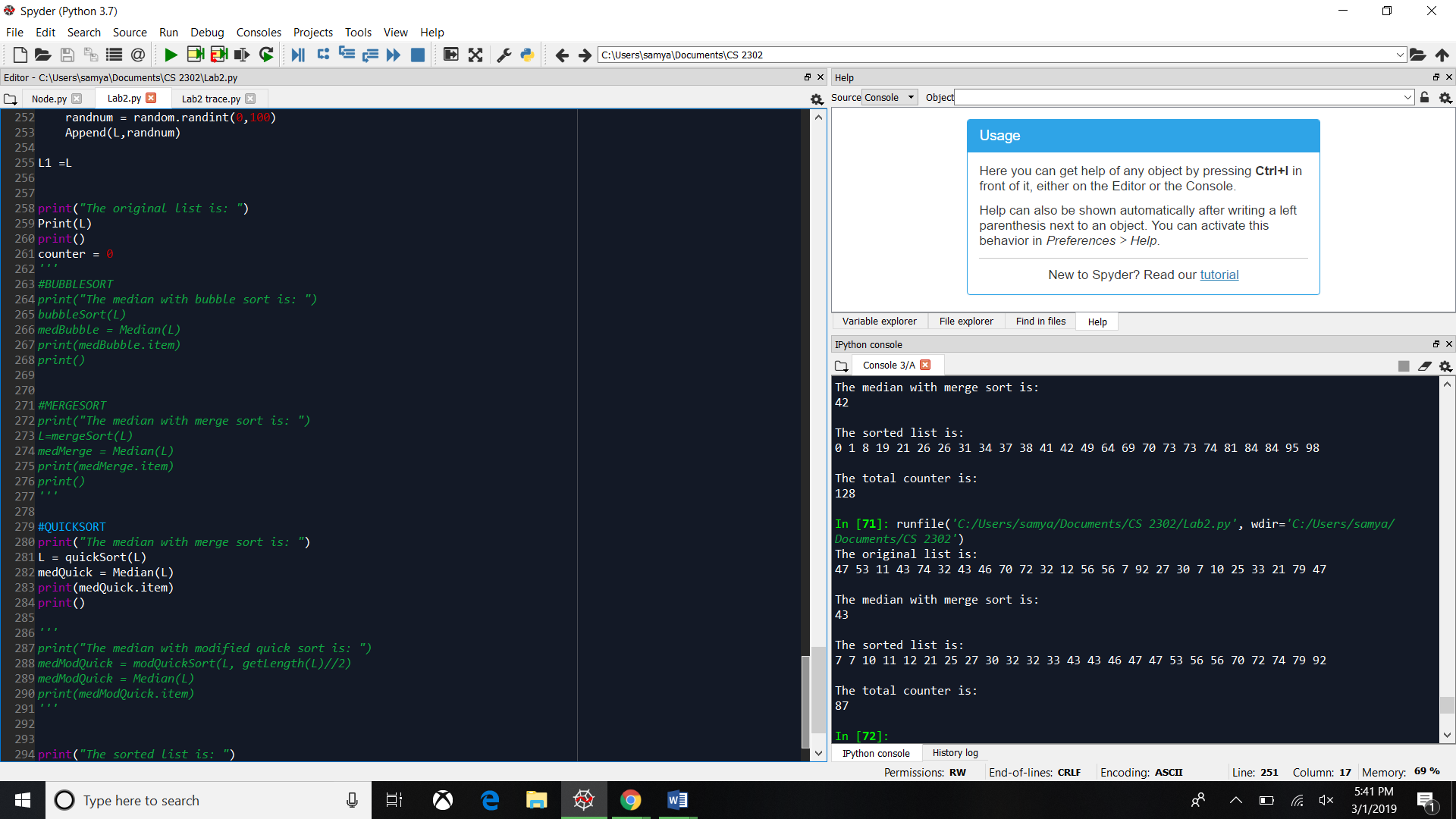
Merge Sort:

This was the toughest to make in my opinion, but in order to do this sorting method I divided the original list into 2 equal parts. I got the middle node and made it point to null so the lists separate. Then I start with the recursion, I use 2 recursion calls one for the left part which is where the smaller numbers going to be and the right side which is where larger numbers go. Since its recursions its going to keep creating smaller lists until the base case is true which is the list’s length is less than 2 or if its empty. When the base case is true then it calls the sortList method where it compares the heads from left list and right list. Which is were you start moving the numbers in order. The big O for Merge Sort is O(nlogn).

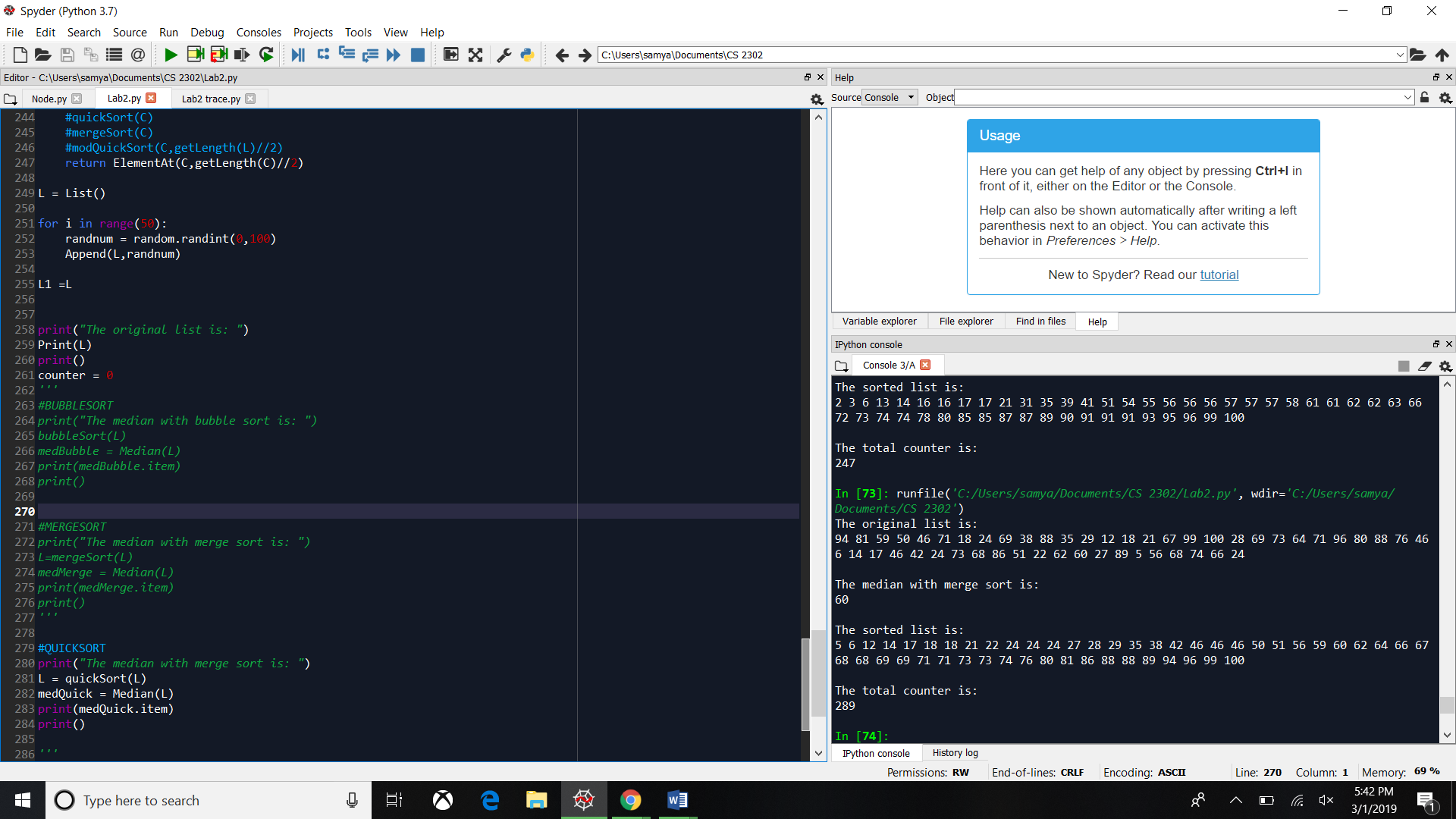
List size: 5



List size: 25

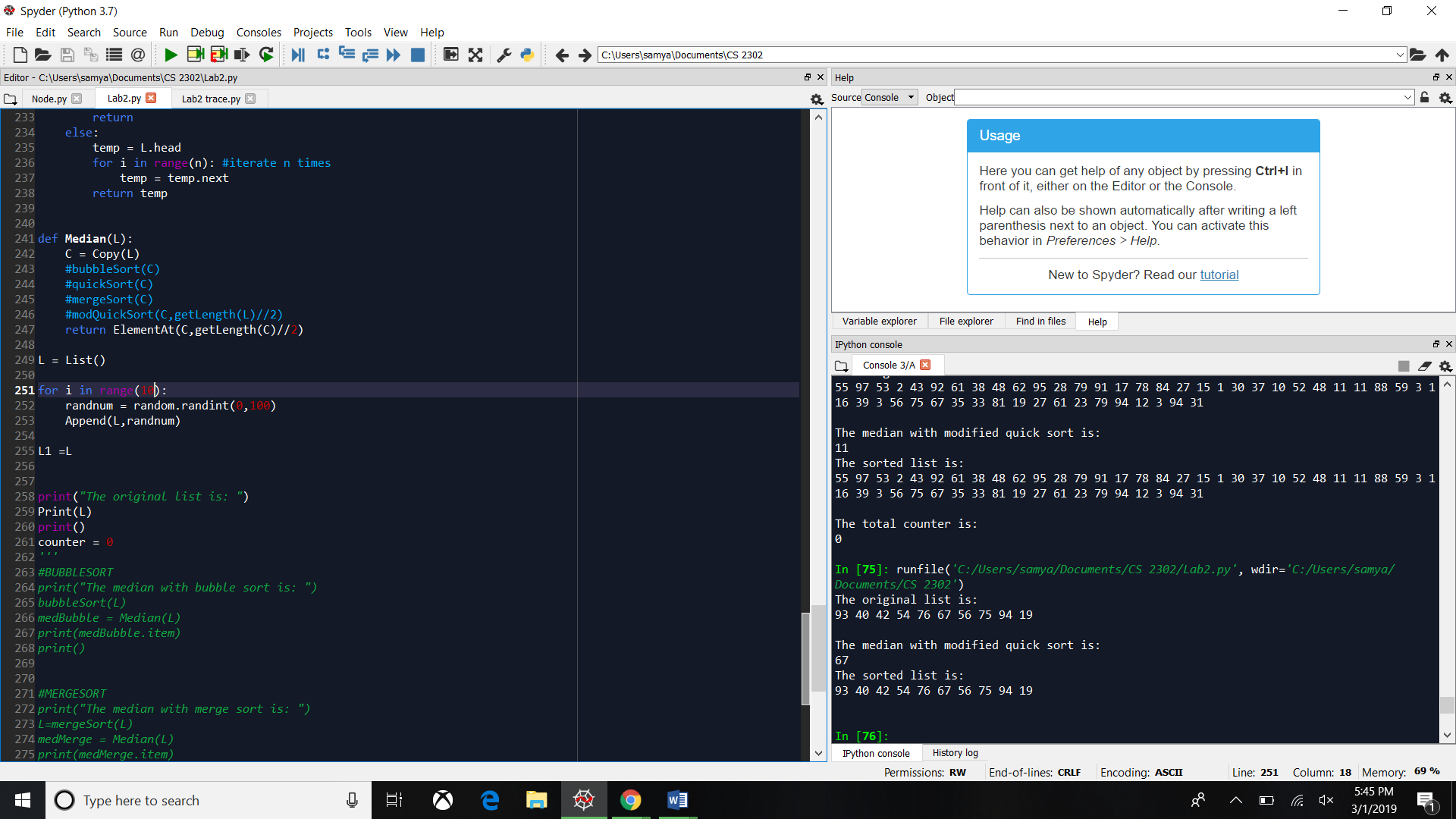


List size: 50



Modified Quick Sort:

Sadly, I was unable for this method to work. It compiled and printed the list, but it never sorted the list.



Time complexity comparison

Bubble Quick Merge

Code:

import random

"""

@author: Samuel Chong

CS2302

Olac Fuentes

Lab 2

Purpose: Sort lists using quick sort, merge sort and bubble sort and getting the median value

"""

class Node(object):

# Constructor

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

self.item = item

self.next = next

def PrintNodes(N):

if N != None:

print(N.item, end=' ')

PrintNodes(N.next)

def PrintNodesReverse(N):

if N != None:

PrintNodesReverse(N.next)

print(N.item, end=' ')

#List Functions

class List(object):

# Constructor

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

self.head = None

self.tail = None

def IsEmpty(L):

return L.head == None

def Append(L,x):

# Inserts x at end of list L

if IsEmpty(L):

L.head = Node(x)

L.tail = L.head

else:

L.tail.next = Node(x)

L.tail = L.tail.next

def Print(L):

# Prints list L's items in order using a loop

temp = L.head

while temp is not None:

print(temp.item, end=' ')

temp = temp.next

print() # New line

def PrintRec(L):

# Prints list L's items in order using recursion

PrintNodes(L.head)

print()

def Remove(L,x):

# Removes x from list L

# It does nothing if x is not in L

if L.head==None:

return

if L.head.item == x:

if L.head == L.tail: # x is the only element in list

L.head = None

L.tail = None

else:

L.head = L.head.next

else:

# Find x

temp = L.head

while temp.next != None and temp.next.item !=x:

temp = temp.next

if temp.next != None: # x was found

if temp.next == L.tail: # x is the last node

L.tail = temp

L.tail.next = None

else:

temp.next = temp.next.next

def PrintReverse(L):

# Prints list L's items in reverse order

PrintNodesReverse(L.head)

print()

def getLength(L):

count = 0

temp = L.head

if L.head == None:

return 0

else:

while temp is not None:

count +=1

temp = temp.next

return count

def bubbleSort(L):

global counter

counter = 0

sort = True

while sort: #while sort is true

temp = L.head

sort = False

while temp.next is not None:

if temp.item > temp.next.item: #if the current item is greater than the next then swap position

t = temp.item

temp.item = temp.next.item

temp.next.item = t

counter +=1

sort = True #if something was swapped, then true

temp = temp.next

def quickSort(L):

global counter

if getLength(L) <= 1: #base case

return L

if getLength(L) > 1:

pivot = L.head.item

temp = L.head.next

L1 = List()

L2 = List()

while(temp is not None): #after pivot

counter += 1

if temp.item < pivot: #if item is less than the pivot, it goes to the small list

Append(L1, temp.item)#insert n in L1

else:

Append(L2, temp.item)#insert n in L2

temp = temp.next

L1 = quickSort(L1)

L2 = quickSort(L2)

Append(L1,pivot)

L = addLists(L1,L2) #merge L1 and L2 into a single list

return L

def addLists(L1,L2): # method to merge the lists for quick sort

if(IsEmpty(L1)): #if list smaller than pivot is empty

return L2

elif(IsEmpty(L2)): #if list greater than pivot is empty

return L1

else:

L1.tail.next = L2.head

L1.tail = L2.tail

return L1

def modQuickSort(L,median):

count = 0

pivot = L.head.item

temp = L.head.next

L1 = List()

L2 = List()

while(temp is not None):

if temp.item < pivot:#if item is less than the pivot, it goes to the small list

Append(L1, temp.item)#insert n in L1

count +=1

else:

Append(L2, temp.item)#insert n in L2

count +=1

temp = temp.next

if getLength(L1) < median: #if median is not in L1, then search in L2

count +=1

return modQuickSort(L2,getLength(L1)-1)

elif getLength(L2) < median: #if the median not in L2

count +=1

return modQuickSort(L1,median)

else: #if it isnt in L1 or L2

return pivot

def sortList(l,r):

global counter

result = None

if l is None or getLength(l) < 1:

return r

if r is None or getLength(r) < 1:

return l

if l.head.item <= r.head.item:

result = l.head

temp = l.head.next

tempL = List()

tempL.head = temp

result.next = sortList(tempL, r).head

counter =+1

else:

result = r.head

temp = r.head.next

tempL = List()

tempL.head = temp

result.next = sortList(l, tempL).head

counter =+1

L = List()

L.head = result

return L

def mergeSort(L):

global counter

if L is None or getLength(L)<2:

return L

mid = ElementAt(L,(getLength(L) // 2)-1)

nextMid = mid.next

mid.next = None

counter =+1

left = mergeSort(L)

R = List()

R.head = nextMid

counter =+1

right = mergeSort(R)

sortedList = sortList(left, right)

return sortedList

def Copy(L):

copyList = List() #create new list

temp = L.head

while temp is not None:

Append(copyList,temp.item) #add the item in temp everytime it iterates in the loop

temp = temp.next

return copyList

def ElementAt(L,n):

if(getLength(L) < n):

return

else:

temp = L.head

for i in range(n): #iterate n times

temp = temp.next

return temp

def Median(L):

C = Copy(L)

#bubbleSort(C)

#quickSort(C)

#mergeSort(C)

#modQuickSort(C,getLength(L)//2)

return ElementAt(C,getLength(C)//2)

L = List()

for i in range(10):

randnum = random.randint(0,100)

Append(L,randnum)

L1 =L

print("The original list is: ")

Print(L)

print()

counter = 0

'''

#BUBBLESORT

print("The median with bubble sort is: ")

bubbleSort(L)

medBubble = Median(L)

print(medBubble.item)

print()

#MERGESORT

print("The median with merge sort is: ")

L=mergeSort(L)

medMerge = Median(L)

print(medMerge.item)

print()

'''

#QUICKSORT

print("The median with quick sort is: ")

L = quickSort(L)

medQuick = Median(L)

print(medQuick.item)

print()

'''

print("The median with modified quick sort is: ")

medModQuick = modQuickSort(L, getLength(L)//2)

medModQuick = Median(L)

print(medModQuick.item)

'''

print("The sorted list is: ")

Print(L)

print()

print("The total counter is: ")

print(counter)

I, Samuel Chong, sign the academic honesty certification. This is my work and only my work. No external help was used for this lab. Also, this report was made by me and no collaboration was made.