**Lab 2 report**

This code has the purpose of sorting through a randomly filled list of numbers in 4 different ways, Bubble Sort, Merge Sort, Quick Sort and Modified Quick Sort. It will provide a comparison in their effectiveness by counting the times items are moved in order to arrange the list in ascending order. The Bubble Sort method is the simplest of them and it simply compares the current element to the one that follows making use of two loops, one to make sure the list is ordered beginning to end and the other used to sort through it the necessary times. Merge Sort is composed of two methods, the main method that divides the list in to equal parts by using recursion and then uses the second method merges the separate parts into a single ordered list (Or is supposed to). Quick Sort makes use of the first element of the list as a pivot, and then using a loop divides the rest of the list in to two new lists that use the pivot as a guide to measure their place in each list, afterwards, the method is called recursively twice in order to sort each of the two lists, then when this is done there are 2 lists, one with elements bigger and one with smaller than the pivot; the pivot is appended to the small list and then the large list is appended to the other one, returning a single ordered list. Lastly the Modified Quick Sort has the same structure of the original, the difference is this one separates, then appends it again in order, afterwards it uses a recursion call to check it again and finally returns an ordered list.

**BubbleSort: O(n2)**

**MergeSort: O(n log n)**

**QuickSort: O(n log n)**

**ModQuickSort: O(n log n)**

**Experimental Results**

**Conclusion**

The most effective sorting method is the modified Quick sort with a O(n log n) it had to do drastically less comparisons than the rest of the sorting methods, however I also had to put some more effort into finding the appropriate way to code it.

**Source Code**

import random

#Node Functions

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):

temp = L.head

count = 0

while temp is not None:

count += 1

temp = temp.next

return count

def RandomGenerator(n):

L = List()

for i in range(n):

x = random.randrange(101)

Append(L,x)

return L

def BubbleSort(L):

count = 0

cond =False

while not cond:

cond = True

temp = L.head

while temp.next is not None:

if temp.item > temp.next.item:

t = temp.item

temp.item = temp.next.item

temp.next.item = t

cond = False

count+=1

temp = temp.next

#print(count)

def MergeSort(L):

size = GetLength(L)

temp = L.head

L1 = List()

L2 = List()

L3 = List()

count = 0

if size > 1:

mid = size//2

for i in range(size):

if i < mid:

Append(L1, temp.item)

temp = temp.next

count += 1

else:

Append(L2,temp.item)

temp = temp.next

count += 1

print(count)

MergeSort(L1)

MergeSort(L2)

L3.head = Merge(L1.head,L2.head)

return L3

def Merge(temp1,temp2):

L = None

if temp1 == None:

return temp1

if temp2 == None:

return temp2

if temp1.item >= temp2.item:

L = temp2

L.next = Merge(temp1,temp2.next)

else:

L = temp1

L.next = Merge(temp1.next, temp2)

return L

def QuickSort(L):

size = GetLength(L)

temp = L.head

count = 0

if size > 1:

pivot = temp.item

temp = temp.next

L1 = List()

L2 = List()

while temp is not None:

if temp.item < pivot:

Append(L1,temp.item)

else:

Append(L2,temp.item)

count +=1

temp = temp.next

print(count)

L1 = QuickSort(L1)

L2 = QuickSort(L2)

Append(L1,pivot)

L = List()

temp2 = L1.head

while temp2 != None:

Append(L, temp2.item)

temp2 = temp2.next

temp3 = L2.head

while temp3 != None:

Append(L, temp3.item)

temp3 = temp3.next

return L

def ModQuickSort(L):

size = GetLength(L)

temp = L.head

if size > 1:

pivot = temp.item

temp = temp.next

L1 = List()

L2 = List()

count = 0

while temp is not None:

if temp.item < pivot:

Append(L1,temp.item)

else:

Append(L2,temp.item)

count += 1

temp = temp.next

Append(L1,pivot)

L = List()

temp2 = L1.head

while temp2 != None:

Append(L, temp2.item)

temp2 = temp2.next

temp3 = L2.head

while temp3 != None:

Append(L, temp3.item)

temp3 = temp3.next

print(count)

L = QuickSort(L)

return L

def Median(L):

temp = L.head

size = GetLength(L)//2

for i in range(size):

temp = temp.next

return temp.item

L = RandomGenerator(6)

Print(L)

#L = MergeSort(L)

#Print(L)

#BubbleSort(L)

#Print(L)

L = QuickSort(L)

Print(L)

#print(Median(L))

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.

\_\_\_\_\_\_\_\_\_\_\_\_Hugo Chavez\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_