**Introduction**

The task to be accomplished required that a linked list be sorted. The methods to be used are Bubble sort, Quick sort, Merge sort, as well as an improved version of Quick sort. The final step for completion is the returning of the middle element of a given sorted-list, afterwards, compute and compare the running times of each method.

**Proposed solution design and implementation**

Understanding the logic behind Bubble sort, the main function behind this method requires that every node’s item in a given linked list is compared to every next node’s item, shifting until every node is sorted in an ascending order. Upon completion, return the middle element, this function should take O(n^2) time to complete.

Attempting Merge sort, it’s understood that the objective was to recursively create two lists of equal size, then sort them in the process, then finally merge them into one complete and sorted list. From here, return the middle element, this function should take O(log n) time to complete.

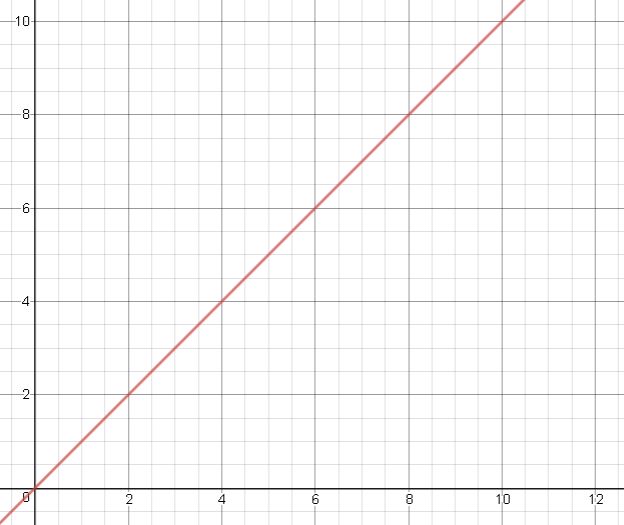
Attempting Quick sort, it’s understood that the objective was to make the head of a given list a pivot, then create two lists, one would house the items from the list that are less than the pivot, the second would house the items greater than the pivot, after this, append the pivot to the end of the first list or prepend to the second one. Finally, concatenate the lists, then return the middle element, this function should take O(log n) time to complete.

Attempting the improved version of Quick sort, not much was understood, the algorithm of Quick sort was not completed successfully, thus the improved version could not be accomplished at all. It was understood that that if the length of both lists were equal, then the middle element would be the pivot, else the middle element would be located in the longer list. The time complexity would have been O(log n) if completed successfully.

**Experimental results**

The objectives of the lab were not met, except for Bubble sort. The following list was generated randomly. List: 6 6 1 3 0. It should be noted that Quick sort and Merge sort both use the Merge function, which is a rendition of Bubble sort.

Expected Time Complexity Graphs:

Bubble

0 1 3 6 6

Times called: 8

Mid 3

Time Complexity: O(n^2)

Merge

0 1 3 6 6

Times called: 9

Mid 3

Expected Time Complexity: O(log n)

Quick

0 1 3 6

Times called: 9

Mid 3

Expected Time Complexity: O(log n)

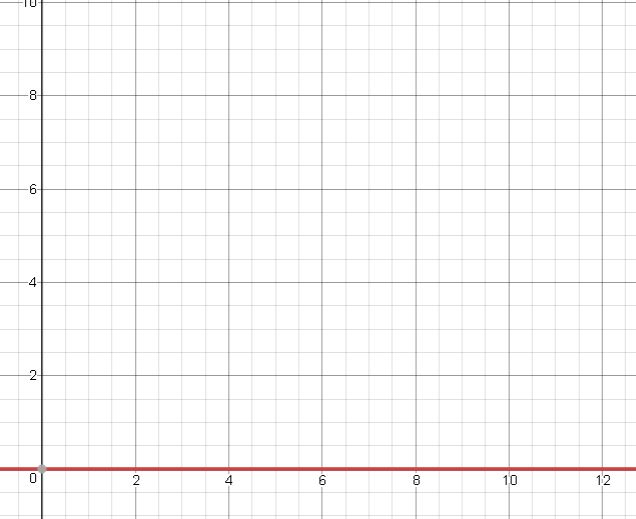
Quick 2

6 6 1 3 0

Times called: 2

Mid 1

Expected Time Complexity: O(log n)

List Length 1

1

Times called: 10

Mid 1

**Conclusions**

Having completed this assignment, the main lesson learned is that not everything goes the way it was planned. Having worked on this, the failure to comprehend Mergesort and Quicksort has led to the understanding that more studying should be done outside of the classroom environment to fully comprehend the objectives of the lectures, as well as the objectives of the labs.

**Appendix**

"""

Course: CS 2302

Author: Jacob Montenegro

Lab: 2

Instructor: Dr. Olac Fuentes

T.A.: Anindita Nath, Maliheh Zargaran

Date of Last Modification: 2/22/2019

Program's Purpose: Sort lists using Bubble sort, Merge sort, Quick sort, and improved Quick sort.

"""

import random

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 Prepend(L,x):

# Inserts x at start of list L

if IsEmpty(L):

L.head = Node(x)

L.tail = L.head

else:

q=Node(x,L.head)

L.head=q

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 GetLength(L):

temp=L.head

count=0

while temp is not None:

count+=1

temp = temp.next

return count

#returns ith element of List

def ElementAt(L,i):

temp = L.head

count=0

while temp is not None:

if count == i:

return temp.item

temp=temp.next

count+=1

return None

#Creates new List with same elements/items of L

def Copy(L):

C=List()

temp=L.head

while temp is not None:

Append(C,temp.item)

temp=temp.next

return C

#counts the number of elements, returns the count

def getLength(L):

temp = L.head

count=0

while temp is not None:

count+=1

temp=temp.next

return count

#Main method for bubble sort

def MedianB(L):

global a

C = Copy(L)

BubbleSort(C)

print('Times called:',a)

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

def BubbleSort(L):

global a

temp=L.head

done = False

while done is not True:

done = True

temp = L.head

while temp is not None:

if temp.next is not None:

if temp.item>temp.next.item:

a+=1

t=temp.item

temp.item=temp.next.item

temp.next.item=t

done=False

temp=temp.next

return L

#main method for quick sort

def MedianQ(L):

global c

C = Copy(L)

l1,r1 = QuickSort(C)

C = Merge(l1,r1)

print('Times called:',c)

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

def QuickSort(L):

global c

c+=1

if getLength(L)>1:

temp=L.head

pivot=L.head.item

L1=List()

R1=List()

while temp is not None:

i=temp.item

if i<pivot:

Append(L1,i)

if i>pivot:

Append(R1,i)

temp=temp.next

Append(L1,pivot)

QuickSort(L1)

QuickSort(R1)

return L1,R1

#main method for merge sort

def MedianM(L):

global b

C = Copy(L)

C=MergeSort(C)

print('Times called:',b)

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

def MergeSort(L):

global b

b+=1

L1=List()

L2=List()

if getLength(L)>1:

mid=getLength(L)//2

end=getLength(L)

temp = L.head

while temp is not None:

for i in range(mid):

Append(L1,temp.item)

temp=temp.next

for j in range(mid,end):

Append(L2,temp.item)

temp=temp.next

MergeSort(L1)

MergeSort(L2)

L=Merge(L1,L2)

return L

def Merge(L1,L2):

temp = L1.head

t = L2.head

List1=List()

a = [None]\*getLength(L1)

b = [None]\*getLength(L2)

i = 0

j = 0

while temp is not None:

a[i]=temp.item

temp = temp.next

i+=1

while t is not None:

b[j]=t.item

t = t.next

j+=1

for i in range(len(a)):

Append(List1,a[i])

for i in range(len(b)):

Append(List1,b[i])

BubbleSort(List1)

return List1

#main method for improved quick sort

def MedianQTwo(L):

global count

C = Copy(L)

QuickSort2(C)

print('Times called:',count)

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

def QuickSort2(L):

global count

if getLength(L)>1:

temp=L.head

pivot=L.head.item

L1=List()

L2=List()

while temp is not None:

i=temp.item

if i<pivot:

Append(L1,i)

if i>pivot:

Append(L2,i)

temp=temp.next

if getLength(L1)>getLength(L2):

count+=1

QuickSort2(L1)

else:

count+=1

QuickSort2(L2)

Prepend(L2,pivot)

t=L1.head

if getLength(L1)==getLength(L2):

return pivot

while t is not None:

Append(L2,t.item)

t=t.next

return L2

L = List()

for i in range(5):

q = random.randint(0,10)

Prepend(L,q)

l=L

print("Length:",GetLength(l))

global a

a=0

print('Bubble')

print('Mid',MedianB(l))

l=L

print()

print('Merge')

global b

b=0

print('Mid',MedianM(l))

print()

l=L

print('Quick')

global c

c=0

print('Mid',MedianQ(l))

print()

l=L

print('Quick 2')

global count

count=0

print('Mid',MedianQTwo(l))

print()

qw=List()

count=0

Append(qw,1)

print('Mid',MedianM(qw))

I certify that this project is entirely my own work. I wrote, debugged, and tested the code 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. Signed, Jacob M. Montenegro