Andres Nava

2-7-19

CS 2302 Data Structures

1:30-2:50

80532614

**Introduction:**

This labs code required us to make random linked list and sort them from least to greatest using different methods of sorting. The sorts we have to make and use are bubble sort, merge sort, and 2 different version of quick sort.

**Solutions:**

So for the bubble sort the way I solved it was make the head item the first and check each other item in the list and switch when it was greater than one of them I also made it a while loop that changes to true once it has gone through the list several time and made sure they are all in there respected places.

For the merge one I made a split method which split the list in half. The way I did that was also make a method to get the length of the list and only do half and get the other half into another list. After that I kept recursively calling it and made a merge sort method which took the item and sort them then I merge the 2 list together and sorted it out.

For the first quick sort I mad e a pivot that pointed to the head and a temp to the next item. I then checked the pivot to the next item and switch if the pivot was bigger than the next item and went on so forth. For the second quick sort I made it only do one recursive call and use the other methods I had to sort and split them up.

**Experimental :**

For the experimental I basically had a bunch of random number generated and made sure my merge worked. After I got merge working I had to figure out a way to get to the halfway point. What I did was move one unit only by one and the other unit by two which made the second unit go faster and reach the end twice as fast thus getting me the halfway point.  
**Conclusion:**

I learned how to use lists and create them and also how to use the nodes

**Appendix:**

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=' ')

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 item at end of list

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

temp = L.head

while temp is not None:

print(temp.item, end=' ')

temp = temp.next

print() # New line

def PrintRec(L):

# Prints 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:

if temp.next == L.tail:

L.tail = temp

L.tail.next = None

else:

temp.next = temp.next.next

def PrintReverse(L):

# Prints items in reverse order

PrintNodesReverse(L.head)

print()

def GetLength(L):

if L is None:

return 0

temp = L.head

count = 0

while temp is not None:

temp = temp.next

count += 1

return count

def Copy(L):

C = List()

if IsEmpty(L):

return C

else:

temp = L.head

while temp is not None:

Append(C,temp.item)

temp = temp.next

return C

def Median(L):

C = Copy(L)

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

def ElementAt(L,x):

count = 0

while L.head is not None:

if count is not x - 1:

L.head = L.head.next

count +=1

else:

return L.head.item

def BubbleSort(L):

#Bubble Sort

#O(n^2)

global count

change = True

count = 0

if IsEmpty(L):

return

while change:

temp = L.head

change = False

while temp.next is not None:

count += 1

if temp.item> temp.next.item:

temp2 = temp.item

temp.item = temp.next.item

temp.next.item = temp2

change = True

temp = temp.next

# print(count)

return L

def QuickSort(L):

global count

if GetLength(L) > 1:

pivot = L.head.item # make s pivot the head

temp = L.head.next

L1 = List()

L2 = List()

count = 0

while temp is not None:

count = count + 1

#splits list by elements that are less or more than the pivot

if temp.item <= pivot:

count = count + 1

Append(L1,temp.item)

else:

Append(L2,temp.item)

count = count + 1

temp = temp.next

#recursive calls to edit the list

L1 = QuickSort(L1)

L2 = QuickSort(L2)

#Adds pivot to the middle

Append(L1, pivot)

#combines lists

return Concatenate(L1,L2)

else:

return L

def Concatenate(L1,L2):

if IsEmpty(L1):

return L2

if IsEmpty(L2):

return L1

L1.tail.next = L2.head

L1.tail = L2.tail

return L1

def ModifiedQuick(L):

if L.head is not None:

pivot = L.head.item

temp = L.head.next

L1, L2= List(), List()

count = 0

#the median will belong in the longer list

while temp is not None:

count += 1

if temp.item <= pivot:

Append(L1, temp.item)

else:

Append(L2, temp.item)

temp = temp.next

if GetLength(L1) > GetLength(L2):

L2 = QuickSort(L1)

return L2

else:

L1 = QuickSort(L2)

return L1

else: return L

def SplitList(L):

temp = L.head

L1 = List()

L2 = List()

n = 0

#Separates lists

while n < GetLength(L)//2:

Append(L1,temp.item)

n = n + 1

temp = temp.next

while n < GetLength(L):

Append(L2,temp.item)

n = n + 1

temp = temp.next

return L1, L2

def MergeSort(L):

if L.head is not None and L.head.next is not None:

L1, L2 = SplitList(L)

L1 = MergeSort(L1)

L2 = MergeSort(L2)

sort = Merge(L1,L2)

return sort

else: return L

def Merge(L1, L2):

global count

sort = List()

count = 0

current = L1.head

current2 = L2.head

#compares two elemnts of the lists, whoever is smallest gets appended

#then the following element of the list gets compared with the one that did not get appended

while current is not None and current2 is not None:

count += 1

if current.item < current2.item:

Append(sort, current.item)

current = current.next

else:

Append(sort, current2.item)

current2 = current2.next

#Appends any left over elements

while current is not None:

Append(sort, current.item)

current = current.next

while current2 is not None:

Append(sort, current2.item)

current2 = current2.next

return sort

def ListFiller(n):

L = List()

for i in range(n):

Append(L,random.randint(0, 101))

return L

# main

L = ListFiller(11)

print("Unsorted List:", end = ' ')

Print(L)

print("Bubble Sort: ", end = ' ')

BubbleSort(L)

Print(L)

print("Median :", end = ' ')

print(Median(L))

print('Count: ', count)

print()

print("Unsorted List:", end = ' ')

Print(L)

print("Merge Sort: ", end = ' ')

MergeSort(L)

Print(L)

print("Median :", end = ' ')

print(Median(L))

print('Count: ', count)

print()

print("Unsorted List:", end = ' ')

Print(L)

print("Quick Sort: ", end = ' ')

QuickSort(L)

Print(L)

print("Median :", end = ' ')

print(Median(L))

print('Count: ', count)

print()

print("Unsorted List:", end = ' ')

Print(L)

print("Modified Quick Sort: ", end = ' ')

ModifiedQuick(L)

Print(L)

print("Median :", end = ' ')

print(Median(L))

print('Count: ', count)

“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.”