**Lab #2**

CS 2302

Ana Luisa Mata Sanchez

# Introduction

This assignment is focused on implementing three sorting algorithms on a Singly Linked List. These algorithms are merge sort, bubble sort, and two different quick sorts. When the program is ran it should sort the list in ascending order and return the item in the median position.

# Proposed solution design and implementation

**Module 1 – Bubble Sort**

The following figures and text explain bubble sort pseudo-code:

When the method is first called we assume that the list is not sorted. We keep track of this with a variable.

isSorted = False

We need a temporary node to iterate through the list.

curr = L.head

We check if the list has only one element or if it is empty, if it is we do nothing

if L.head is None or curr.next is L.tail:

return L

We now check to see if the list is Sorted or not.

while isSorted is false:

We now assume that this pass will result in the list being sorted.

isSorted = true

If it is not we must traverse the whole list from 0 to the length – 1

for i in range(GetLength(L)-1):

Now we check two adjacent items making sure that both are not None, if the first is bigger we exchange it with the second, effectively sorting them. We will keep comparing until we reach the last element in the list.

if curr.item > curr.next.item and curr.next is not None and curr is not None:

temp = curr.next.item

curr.next.item = curr.item

curr.item = temp

If we swap we must tell the program that we must have another pass and keep iterating outside the if statement using our temporary node.

isSorted = false

curr = curr.next

The loop that checks if it’s sorted will finally end when no swaps are made, thus the list will be sorted.

**Module 2 – Merge Sort**

The following figures and text explain merge sort pseudo-code:

Every time we check to see that the list has more than one element

if GetLength(L)>1

We create two lists so that we can sort each one.

A = List()

B = List()

We create a temp variable to iterate.

temp = L.head

Now we separate in two different list exactly in half.

for i in range(half)

Append(A,temp.item)

temp = temp.next

while temp!= None

Append(B,temp.item)

temp = temp.next

Then we keep splitting in two until we reach the last element.

MergeSort(A)

MergeSort(B)

After splitting in two we have to merge in order. I use a separate method.

L.head = Merge(A, B).head

In the merge method we will need to variables to iterate through both lists.

left = A.head

right = B.head

We will keep comparing until we reach the end of either list

while left is not None and right is not None

If the left list item is less than the item on the right list you append the smallest and advance that list. Otherwise, do the opposite.

while left is not None and right is not None

if left.item < right.item

Append(L, left.item)

left = left.next

else

Append(L, right.item)

right = right.next

Since the lists are not always evenly distributed, there might be items left on either list. We must iterate through the list that has remaining elements and add them to the final list.

if right is None

while left is not None

Append(L, left.item)

left = left.next

else if left is None

while right is not None

Append(L, right.item)

right = right.next

**Module 3 - Quick Sort**

The following figures and text explain quicksort pseudo-code:

Every time we check to see that the list has more than one element

if GetLength(L)>1

We pick a pivot. This will decide how we split each list. In this case it is easy to pick the head because we always have a pointer to it.

pivot = L.head.item

We create two lists so that we can sort each one.

A = List()

B = List()

We create a temp variable to iterate that is not the same as our pivot.

temp = L.head.next

Now we separate in two different using the pivot. If the number is bigger than the pivot we put it on the right list, and if it is smaller we put it in the left list.

while temp is not None

if temp.item>pivot

Append(right, temp.item)

else

Append(left, temp.item)

temp = temp.next

Then we keep picking a new pivot and splitting accordingly.

QuickSort(left)

QuickSort(right)

After we are done splitting, we must append the pivot that we did not include to either side. I chose to append to the left.

Append(left, pivot)

Then we have to connect the left and right together, making one list.

left.tail.next = right.head

Then to change the values in the original list we must evaluate three cases: both the head of the left list and the tail of the right list are not none, the head of the left list is none, and, the head of the right list is None.

if left.head is not None and right.tail is not None

L.head = left.head

L.tail = right.tail

Else if left.head is None

L.head = right.head

L.tail = right.tail

Else if right.head is None

L.head = left.head

L.tail = left.tail

**Module 4 – Modified Quick Sort**

The following figures and text explain the modified quicksort pseudo-code:

Every time we check to see that the list has more than one element, if it does then we continue

if GetLength(L)>1

We pick a pivot. This will decide how we split each list. In this case it is easy to pick the head because we always have a pointer to it.

pivot = L.head.item

We create two lists so that we can sort each one.

left = List()

right = List()

We create a temp variable to iterate that is not the same as our pivot.

temp = L.head.next

Now we separate in two different using the pivot. If the number is bigger than the pivot we put it on the right list, and if it is smaller we put it in the left list.

while temp is not None

if temp.item>pivot

Append(right, temp.item)

else

Append(left, temp.item)

temp = temp.next

Different to the normal quicksort,

QuickSort(left)

QuickSort(right)

After we are done splitting, we must append the pivot that we did not include to either side. I chose to append to the left.

Append(left, pivot)

Then we have to connect the left and right together, making one list.

left.tail.next = right.head

Then to change the values in the original list we must evaluate three cases: both the head of the left list and the tail of the right list are not none, the head of the left list is none, and, the head of the right list is None.

if left.head is not None and right.tail is not None

L.head = left.head

L.tail = right.tail

Else if left.head is None

L.head = right.head

L.tail = right.tail

Else if right.head is None

L.head = left.head

L.tail = left.tail

# Experimental results

**Bubble Sort**

Big-O running time:

**Modified code:**

def BubbleSort(L):

#boolean to keep track if its sorted

count = 0

isSorted = False

curr = L.head

#if empty or only one element, no need to sort

if L.head == None or L.head == L.tail:

return L

#onlyl sorts if it isnt sorted

while isSorted == False:

curr = L.head

#if it never reaches the if statement, it stays true... it did not swap any elements

isSorted = True

for i in range(GetLength(L)-1):

#Compare adjacent elements, if the first is larger than the second then swap

if curr.item > curr.next.item and curr.next!=None and curr!=None:

temp = curr.next.item

curr.next.item = curr.item

curr.item = temp

#since there has been a swap, we have to sort again

isSorted = False

curr = curr.next

count = count + 1

count = count + 1

print(count)

return L

**Method call:**

############### Bubble Sort ###############

print("\*\*\*\*\*\*\*\*\*\*\*\*\*Bubble Sort\*\*\*\*\*\*\*\*\*\*\*\*")

for i in range(13):

L = List()

L = NewRandList(n)

BubbleSort(A)

**See Excel document in same repository for plots and data collected**

**Bubble Sort**

Big-O running time:

**Method call:**

print("\*\*\*\*\*\*\*\*\*\*\*\*\*Bubble Sort\*\*\*\*\*\*\*\*\*\*\*\*")

for i in range(13):

L = List()

L = NewRandList(n)

BubbleSort(L)

**See Excel document in same repository for plots and data collected**

**Merge Sort**

Big-O running time:

T(n) = 2T(n/2) + n

a = 2, b = 2, k = 1

Using the master method…

**Modified code:**

def Merge(A, B):

L = List()

left = A.head

right = B.head

while left!=None and right!=None:

if left.item < right.item:

Append(L, left.item)

left = left.next

else:

Append(L, right.item)

right = right.next

print('a', end="")

if right == None:

while left != None:

Append(L, left.item)

left = left.next

elif left == None:

while right != None:

Append(L, right.item)

right = right.next

return L

**Method call:**

print("\*\*\*\*\*\*\*\*\*\*\*\*\*Merge Sort\*\*\*\*\*\*\*\*\*\*\*\*\*")

for i in range(13):

L = List()

L = NewRandList(n)

MergeSort(L)

print()

**See Excel document in same repository for plots and data collected**

**Quick Sort**

Big-O running time:

T(n) = 2T(n/2) + n

a =2, b = 2, k = 1

Using the master method…

**Modified code:**

def QuickSort(L):

if GetLength(L)>1:

#Pick the pivot

pivot = L.head.item

left = List()

right = List()

temp = L.head.next

while temp!=None:

if temp.item>pivot:

Append(right,temp.item)

else:

Append(left,temp.item)

temp = temp.next

print("a",end="")

QuickSort(left)

QuickSort(right)

Append(left, pivot)

left.tail.next = right.head

if left.head!=None and right.tail!=None:

L.head = left.head

L.tail = right.tail

elif left.head == None:

L.head = right.head

L.tail = right.tail

elif right.head == None:

L.head = left.head

L.tail = left.tail

**Method call:**

print("\*\*\*\*\*\*\*\*\*\*\*\*\*Quick Sort\*\*\*\*\*\*\*\*\*\*\*\*\*")

for i in range(13):

L = List()

L = NewRandList(n)

QuickSort(L)

print()

**See Excel document in same repository for plots and data collected**

**Modified Quick Sort**

Big-O running time:

T(n) = T(n/2) + n

a =1, b = 2, k = 1

Using the master method…

**Modified code:**

def ModifiedQuickSort(L,n):

if GetLength(L)>1:

pivot = L.head.item

left = List()

right = List()

temp = L.head.next

while temp!=None:

if temp.item>pivot:

Append(right,temp.item)

else:

Append(left,temp.item)

temp = temp.next

print("c",end="")

if GetLength(left) < n :

return ModifiedQuickSort(right, n-GetLength(left)-1)

elif GetLength(left) == n:

return pivot

elif GetLength(left) > n:

return ModifiedQuickSort(left, n)

else:

return L.head.item

**Method call:**

print("\*\*\*\*\*\*\*\*\*\*\*\*\*Modified Quick Sort\*\*\*\*\*\*\*\*\*\*\*\*\*")

for i in range(13):

L = List()

L = NewRandList(n)

ModifiedQuickSort(L)

print()

**See Excel document in same repository for plots and data collected**

# Conclusion

During this lab I learned how to implement sorting methods on a Singly Linked List and how to calculate their running times.

# Appendix

# Author: Ana Luisa Mata Sanchez

# Course: CS2302

# Assignment: Lab #2

# Instructor: Olac Fuentes

# Description: Implement several algorithms for finding the median of a list of integers.

# T.A.: Anindita Nath

# Last modified: 02/22/2019

# Purpose: Use bubble sort, merge sort, and two types of quick sort

import random

#Node Functions

class Node(object):

# Constructor

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

self.item = item

self.next = next

#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 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 GetLength(L):

temp = L.head

count = 0

#traverse and count each element traversed

while temp is not None:

count += 1

temp = temp.next

return count

def NewRandList(n):

#Creates a new list of size n with random numbers between 0 and n+100

L = List()

for i in range(n):

Append(L,random.randint(0,n+100))

return L

def BubbleSort(L):

#boolean to keep track if its sorted

isSorted = False

curr = L.head

#if empty or only one element, no need to sort

if L.head == None or L.head == L.tail:

return L

#onlyl sorts if it isnt sorted

while isSorted == False:

curr = L.head

#if it never reaches the if statement, it stays true... it did not swap any elements

isSorted = True

for i in range(GetLength(L)-1):

#Compare adjacent elements, if the first is larger than the second then swap

if curr.item > curr.next.item and curr.next!=None and curr!=None:

temp = curr.next.item

curr.next.item = curr.item

curr.item = temp

#since there has been a swap, we have to sort again

isSorted = False

curr = curr.next

return L

def MergeSort(L):

if GetLength(L)>1:

#create a left and right

A = List()

B = List()

half = GetLength(L)//2

temp = L.head

#Sepparate in two, first loop goes from element 0 to the half

for i in range(half):

Append(A,temp.item)

temp = temp.next

#Second loop goes from one after the half to the end

while temp!= None:

Append(B,temp.item)

temp = temp.next

#keep separating until there is only one element

MergeSort(A)

MergeSort(B)

#Merge them and make that the new list

L.head = Merge(A, B).head

return L

def Merge(A, B):

#new list that will contain left and right

L = List()

#temp variables for iterating

left = A.head

right = B.head

while left!=None and right!=None:

#compare the items in left an right, if left is smaller append it and advance the left

if left.item < right.item:

Append(L, left.item)

left = left.next

#if not, append the right element and advance the right pointer, thus you will append from smallest to biggest

else:

Append(L, right.item)

right = right.next

#Checking if any elements were left in either of the lists

if right == None:

while left != None:

Append(L, left.item)

left = left.next

elif left == None:

while right != None:

Append(L, right.item)

right = right.next

return L

def QuickSort(L):

if GetLength(L)>1:

#Pick the pivot

pivot = L.head.item

#create a left and right from the pivot

left = List()

right = List()

temp = L.head.next

while temp!=None:

#if it is bigger than the pivot put it on right

if temp.item>pivot:

Append(right,temp.item)

#if it is smaller put it on the left

else:

Append(left,temp.item)

#keep going until you reach the end of the list

temp = temp.next

#Now split in two again

QuickSort(left)

QuickSort(right)

#Add the pivot that wasn't included

Append(left, pivot)

#Merge from left to right

left.tail.next = right.head

#Now put that merged list into the original list to sort

#if both lists have things, the head is the left and the tail is the right

if left.head!=None and right.tail!=None:

L.head = left.head

L.tail = right.tail

#if left is none then right is the new list

elif left.head == None:

L.head = right.head

L.tail = right.tail

#if right is none then left is the new list

elif right.head == None:

L.head = left.head

L.tail = left.tail

return L

def ModifiedQuickSort(L,n):

if GetLength(L)>=1:

#pick the pivot

pivot = L.head.item

#create lists for left and right

left = List()

right = List()

temp = L.head.next

while temp!=None:

#if it is bigger than the pivot put it on right

if temp.item>pivot:

Append(right,temp.item)

#if it is smaller put it on the left

else:

Append(left,temp.item)

#keep going until you reach the end of the list

temp = temp.next

#if the median is in the right, "sort" the right

if GetLength(left) < n :

return ModifiedQuickSort(right, n-GetLength(left)-1)

#if the median is the pivot

elif GetLength(left) == n:

return pivot

#if the median is in the left, "sort" the left

elif GetLength(left) > n:

return ModifiedQuickSort(left, n)

else:

return L.head.item

def Median(L):

#if the list is empty

if L.head is None:

return None

else:

temp = L.head

count = 0

#traverse the list and count until you reach the middle

while temp!=None and count!=GetLength(L)//2:

count+=1

temp = temp.next

return temp

def Copy(L):

#new list

N = List()

temp = L.head

while temp!=None:

#manually copy each number of the old list

Append(N, temp.item)

temp = temp.next

#return the new copy

return N

L = List()

L = NewRandList(5)

A = Copy(L)

B = Copy(L)

C = Copy(L)

############### Bubble Sort ###############

print("\*\*\*\*\*\*\*\*\*\*\*\*\*Bubble Sort\*\*\*\*\*\*\*\*\*\*\*\*")

print("Original List: ")

Print(L)

A = BubbleSort(A)

print("Median: ")

print(Median(A).item)

print()

############### Merge Sort ###############

print("\*\*\*\*\*\*\*\*\*\*\*\*\*Merge Sort\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Original List: ")

Print(L)

B = MergeSort(B)

print("Median: ")

print(Median(B).item)

print()

############### Quick Sort ###############

print("\*\*\*\*\*\*\*\*\*\*\*\*\*Quick Sort\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Original List: ")

Print(L)

C = QuickSort(C)

print("Median: ")

print(Median(C).item)

print()

############### Modified Quick Sort ###############

print("\*\*\*\*\*\*\*\*\*Modified Quick Sort\*\*\*\*\*\*\*\*")

print("Original List: ")

Print(L)

print("Median: ")

print(ModifiedQuickSort(L, GetLength(L)//2))

print()