Lab 2 Report

The purpose of this lab was to sort a linked list by using bubble sort, merge sort, quick sort and a modified quick sort in which it finds the median faster. After implementing the sort algorithms we were supposed to compare time efficiency between the algorithms for it to sort the list and respectively find the median of the list.

The approach for bubble sort was to simply compare the first two elements in the list then the next two until you reach the end of the list. You repeat this process as long as a switch was made if not the list is sorted.

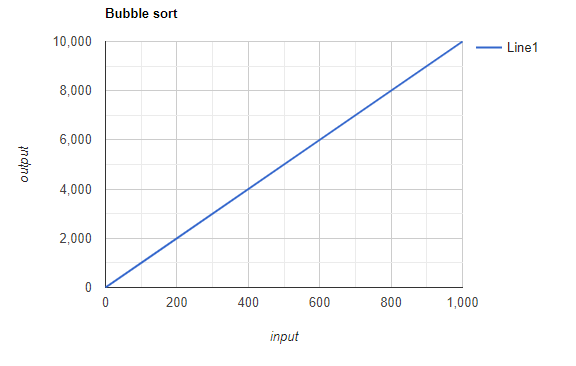
The approach for merge sort I had was to copy the list given divide it into left and right until the lists reach 1 then combine them by comparing which is greater or less and concatenate them till it is sorted.

The approach for quick sort was to pick the head as a pivot and compare it to the next item if it was less prepend it and remove it from the old spot as well it is and if it is greater than append to end of list. Then copy the left of the pivot and recursive call quick sort on it do the same for right until size of 1

Then concatenate lists after comparing less than equal or greater than equal.

The modified version was to go do the first pivot check then depending on if the median was in the left you only check the left if it was in the middle you return pivot and if it was in the right you only check the left so on.

Bubble sort is O(n^2)



Academic Certification statement

I certify that the attached written or electronically transmitted material is my own work. I further certify that (a) I have formally cited or otherwise fully acknowledged the quotations, ideas, and wording used here from other sources, whether published or unpublished, in written or electronic form, (b) I have engaged in no falsification or misrepresentation of data or experience in this submission, (c) I have disclosed any collaboration not specifically authorized by the instructor.

I understand that the relative autonomy of the legal profession carries with it special obligations of self-regulation as outlined in the Columbia Law School Bulletin and that any violation of this certification will subject me to discipline, including possible suspension or expulsion by the Law School, declination to certify for admission to the Bar, and sharing of information about discipline with Bar admissions committees. I further understand that the Law School is required to notify Bar admissions committees if a student has been the subject of disciplinary proceedings, regardless of the outcome of those proceedings.

Justin Ruiloba 2/27/2019

Apendix

#Justin Ruiloba

#2/22/19

#CS 2302

#Lab 2

#Sort single linked list

#using Bubble Sort, Merge Sort, and QuickSort

#as well as return median of each

#imports the random function

import random

#Node Functions

class Node(object):

# Constructor

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

self.item = item

self.next = next

#prints nodes item

def PrintNodes(N):

if N != None:

print(N.item, end=' ')

PrintNodes(N.next)

#Prints Nodes in Reverse

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

#checks if list is empty

def IsEmpty(L):

return L.head == None

#adds Node to end of list updates pointer

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

#adds Node to begginning of List

def Prepend(L,x):

if IsEmpty(L):

L.head = Node(x)

L.tail = L.head

else:

L.head = Node(x,L.head)

#creates a linked List With size of n and random items

def CreateLL(LL,n):

#intial add head node

count=0

#add head node first time

if count == 0:

#Node is equal to head

LL.head = Node(random.randrange(101))

#node is equal to tail

LL.tail = LL.head

#update so conditon does not repeat

count+=1

#Lowers size by one

n-=1

#every case but first time

if count >0:

#while nodes still need to be created

while n>0:

#append to Head

LL.tail.next = Node(random.randrange(101))

LL.tail= LL.tail.next

#reduce by n-1

n-=1

return LL

#prints list

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

#returns item at length node

def ElementAt(LL,length):

#if node is empty return None

if IsEmpty(LL):

return None

#else iterate list

else:

temp = LL.head

#while lis is not empty

while temp is not None:

#if length is == to 0 you have reached destination

if length == 0:

#return the item

return temp.item

#else

#go to next item decrease length by one

length-=1

temp = temp.next

#Not in list

return None

#returns middle element if sorted is median

def median(LL):

return ElementAt(LL,Getlength(LL)//2)

#print Recursive

def PrintRec(L):

# Prints list L's items in order using recursion

PrintNodes(L.head)

print()

#removes Node

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

#search for item in list return the item

def Search(L,item):

if IsEmpty(L):

return None

else:

temp = L.head

while temp is not None:

if temp.item == item:

return item

temp = temp.next

return None

#Bubble sort

def BubbleSort(L,A):

#boolean to see if we swapped

swapcheck = True

#if swapped keep sorting

while swapcheck:

#sets to false if their is no switches exit

swapcheck = False

#starts at begging of list sorts again

A=L.head

#while second item is not Noned compare two items

while A.next is not None:

#if first item is greater swap them

if A.item > A.next.item:

#temp variable to hold value

temp = A.item

#Swap items

A.item = A.next.item

A.next.item = temp

#swap is true

swapcheck=True

#go to next index

A=A.next

#attempt to creat left sublist

def CopyLeft(L,Left,length):

if IsEmpty(L):

return Left

#get list to traverse

Left.head=L.head

Left.tail =L.tail

temp=Left.head

#get to tail of left list

while length>1:

temp =temp.next

length-=1

#print(temp.item)

#tail is equal to node

Left.tail=temp

#left tail is equal to None

Left.tail.next=None

#Print(Left)

#Print(L)

return Left

#Right sublist copy

def CopyRight(L,Right,length):

if IsEmpty(L):

return Right

#set list to transverse

Right.head=L.head

Right.tail =L.tail

temp=Right.head

#Tranverse to new head

while length>0:

temp =temp.next

length-=1

#head is now temp

Right.head=temp

#Print(Right)

#Print(L)

return Right

#attempt to merge list when only size one for merge sort

def m(Left, Right):

#if left is less than right sublist add right.head to left.head.next

if Left.head.item <= Right.head.item:

Left.tail=Right.tail

#if Right is less then left sublist

if Right.item < Left.item:

#Right.tail.next = left.head

#Right.tail = left.tail

temp.head = Left.head

Left.head = Right.head

Right.tail.next = temp.head

#return List

#mergesort attempt trying to break down into left right sublists

def mergeSort(L,Left,Right):

if Getlength(L) <= 1:

return L

#break down left and right further till length 1

mergeSort(L,CopyRight(L,Left,Getlength(L)//2),CopyLeft(L,Left,Getlength(L)//2))

#combine sublists

m(Left,Right)

#quick sort attempt

def QuickSort(L,Left,Right):

if IsEmpty(L):

return L

#if GetLength(L)==1:

#index total of elements to comapre to pivot

index=Getlength(L)

#loop condition

Swapchange=True

while Swapchange:

#pivot always first element

Pivot=L.head.item

#element to comare pivot too

temp=L.head.next

#pivot compared to all elements

if index<=0:

Swapchange=False

#If element is less than or equal to pivot

if temp.item <=Pivot:

#temp variable

a=temp.item

#add item to left sublist

Append(Left,a)

#remove item

Remove(L,a)

#and shift it to left side of pivot

Prepend(L,a)

#if item is greater than pivot

if temp.item > Pivot:

#temp var

a=temp.item

#add to right sublist

Append(Right,a)

#remove from list

Remove(L,a)

#add to right of pivot

Append(L,a)

#update index or amount of elements compared

index-=1

#go to next element to compare

temp=temp.next

#quick sort left and righ lists recursivley

#QuickSort(Left,Left,Right)

#QuickSort(Right,Left,Right)

#get total length of list

def Getlength(L):

if IsEmpty(L):

return 0

else:

temp = L.head

length = 0

while temp is not None:

length +=1

temp=temp.next

return length

#initialize lists used

Right =List()

Left = List()

L = List()

A = List()

#B= List()

#print(IsEmpty(L))

#for i in range(5):

# Append(L,i)

#Prepend(L,100)

#print("search result",Search(L,4))

#print("length is equal to",Getlength(L))

#Print(L)

#creates Lists random values and size

LL=List()

CreateLL(LL,10)

CreateLL(L,5)

#bubble sort

print("Orignal Linked list is")

Print(LL)

BubbleSort(LL,A)

print("BubbleSorted list is")

Print(LL)

print("Median of bubble sort is",median(LL))

#quick sort of

print("Orignal Linked list is")

Print(L)

print("Quick sort is")

QuickSort(L,Left,Right)

Print(L)

print ("Median is",median(L))

#and mergeSort

#mergeSort(L,Left,Right)

#print("merge sort is")

#Print(L)

#print ("Median is",median(L))

#B=CopyRight(L,Right,3)

#Print(B)

#A=CopyLeft(L,Left,3)

#Print(A)

#Print(L)