CS3 Report Lab Two

Daniela Flores

Intro

Lab one consisted of sorting a non-native python list with various sorting algorithms. We had to implement Quick sort, Bubble Sort, Merge Sort and a modified version of Quick sort. After sorting the list with the previously mentioned sorting algorithms, we had to find the median of that list as well.

Proposed Solution Design and Implementation

Before I started writing the methods that would execute the different types of sorting algorithms, I first started by writing a method that would populate a List with random integers. I also wrote a method that would prepend an item into the list when called, a method that would copy the elements in a given List into another new List and another method which would get the length of a given List with a counter. The methods mentioned were vital to correctly execute the sorting algorithms appropriately. After the implementation of the methods mentioned earlier, I went with coding the sorting algorithms. The bubble sort method consisted of a temp variable which stored the head of the List and a while loop which iterated through the given List and sorted it through an if statement that compared the current item with the next item and switched them if the size of the item was larger than the next item.

The method for quick sort consisted of a variable which stored the pivot, in my method the pivot is always the head of the List. After two additional Lists where added for later use. A temp variable was then initialized and the node after the head was stored, as the head was already being used as the pivot. A while loop was used to iterate though the List, inside it an if statement checked that if the item of the node was greater than the pivot it would be stored in one of the lists which was initialized at the beginning of the method. Else the item was appended into the other list. After the while loop was done, a recursive call was made twice with the parameters being the two lists which were being populated in the while loop. At the end of the method the pivot was added back at its correct position and the Lists were reconnected.

I implemented two methods to execute merge sort, on one of these methods I first checked that the Lists length was bigger than one to prevent an error message. Two Lists where then initialized to store half the elements each of the original List. Two recursive calls where then used with the parameters being the Lists with half the original elements. Afterwards another method was called which merged the split lists together into the correct order.

In my main call I first create a List then populate it with random integers. Afterwards I copied the list to three other Lists to correctly compare the results of each sorting algorithm. Each List was then sorted using the sorting methods. The Lists already sorted where then referenced to the method which found the median of given List.

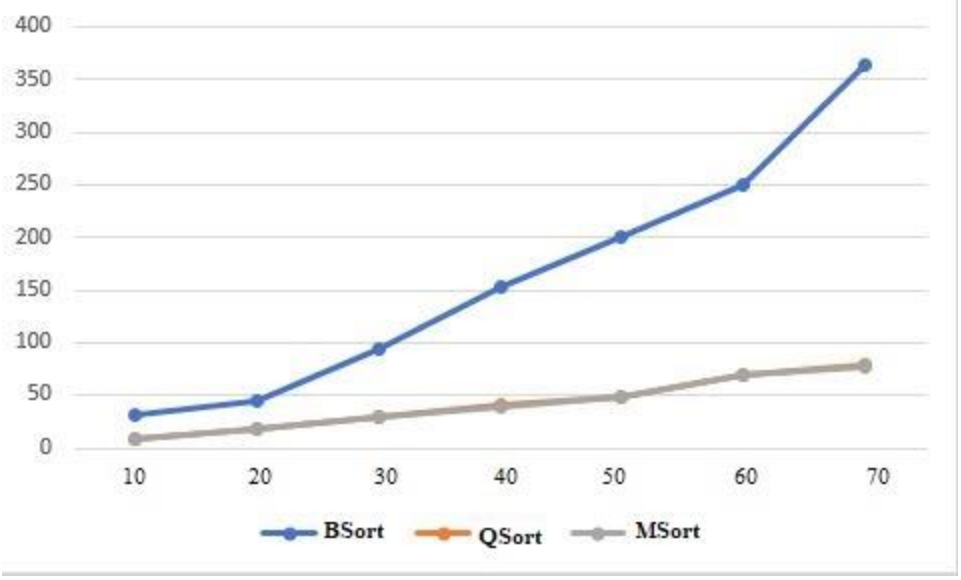
Experimental Results:

I experimented with my methods by changing the length of the List which needed to be sorted. I also had to experiment with my sorting algorithms as I encountered multiple times an error such as 'List has no attribute next' which I discovered was occurring because the List I was passing was either of length one or was empty. In order to prevent those errors, I had to put an if statement at the very beginning of the sorting methods that checked that the List being passed was long enough to be sorted. The following image is what my program outputs. (every time the program is ran, the integers in the List change randomly.)

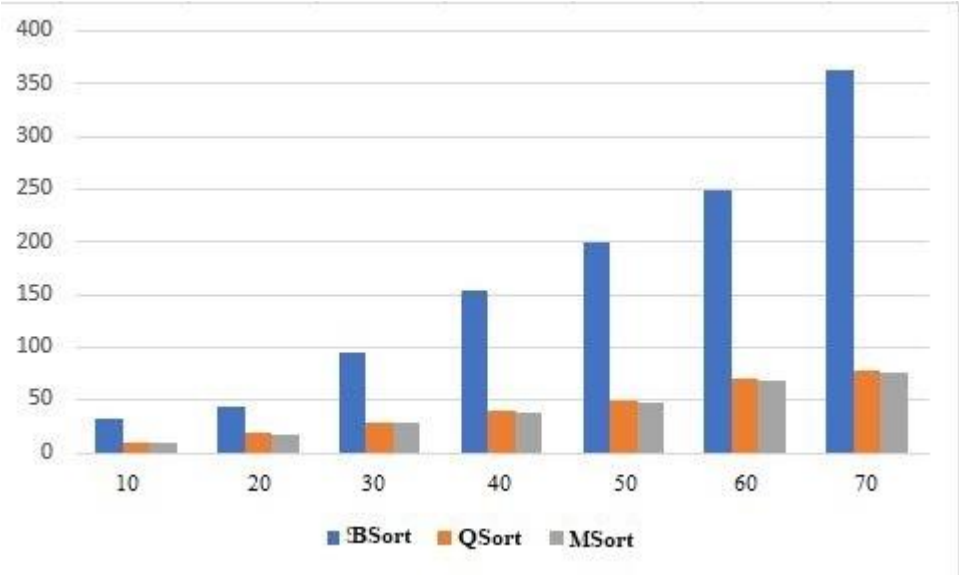
```
In [1]: runfile('C:/Users/flore/.spyder-py3/cs3Lab2.py',
wdir='C:/Users/flore/.spyder-py3')
original list:
70 30 18 36 81 20 1 52 11 34

list sorted though bubble sort:
1 11 18 20 30 34 36 52 70 81
middle element of Linked List sorted by BubbleSort 30

list sorted though Quick sort:
1 11 18 20 30 34 36 52 70 81
middle element of Linked List sorted by QuickSort 30

list sorted though merge sort:
1 11 18 20 30 34 36 52 70 81
middle element of Linked List sorted by merge Sort 30
```

Experimental Results:

Below are the running times of all three sorting algorithms given the same list of same length. X axis represents the length of the list and the Y axis the number of comparisons each sorting algorithm did.



(Quick sort and merge sort did almost the same amount of comparisons that their lines overlap.)



Conclusion:

What I learned from this project was how to traverse or work with a linked list in python. I also learned the running times of the sorting algorithms I had to program. I learned how the sorting algorithms work as I had to write methods which followed that algorithm. I also learned more about python's syntax and what some errors displayed by python meant and what to do to work around them.

Appendix (Source Code):

```
#Course:CS2302
#aAuthor: Daniela Flores
#Lab2
#Instructor: Olac Fuentes
#T.A: Dita Nath
#date of last Mod: 2/26/19
#purpose of program: in this program I had to write methods that'll sort a nonnative python list. I had to implement
#bubble sort, quick sort, merge sort and a modified version of quick sort
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
     #inserts element given at the beginning of linked list
def Prepend(L,x):
    if IsEmpty(L):
        L.head = Node(x)
        L.tail = L.head
    else:
        L.head=Node(x,L.head)

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()
    # empties L to en empty list
def emptying(L):

    L.head = None
    L.tail = None
################################################################
def bubbleSort(L):
    #O(n^2)
    change = True
    while change:
        t = L
        t = L.head
        change = False
        count = 0
        while t.next is not None:
            count +=1
```

```
            if t.item > t.next.item:
                temp = t.item
                t.item = t.next.item
                t.next.item = temp
                change = True
            t = t.next
############################################################
def mergeSort(L):
    #checks if list is long enough
    if getLength(L) > 1:
        p1 = List()
        p2 = List()
        t = L.head
        #splits list into halves
        for i in range(getLength(L) //2):
            Append(p1, t.item)
            t=t.next
        while t != None:
            Append(p2, t.item)
            t = t.next
        #merge sorts the two halves
        mergeSort(p1)
        mergeSort(p2)
        #restores L to an empty list
        emptying(L)
        #puts together lists into L
        mergeTgthr(L, p1, p2)
```

```
def mergeTgthr(L,p1, p2):
    #temp var for split lists
    firstH = p1.head
    secH = p2.head
    #brings together list in order by comparing elements of both split lists
    while firstH != None and secH != None:
        if firstH.item < secH.item:
            Append(L, firstH.item)
            firstH = firstH.next
        else:
            Append(L, secH.item)
            secH = secH.next
    # if statements cover if the halves arent of same length
    if secH is None:
        while firstH != None:
            Append(L, firstH.item)
            firstH = firstH.next
    if firstH is None:
        while secH != None:
            Append(L, secH.item)
            secH = secH.next
#################################################################
# populates list of size n with random intergers
def randomListGen(n):
```

```
    #i = 0
    R = List()
    for i in range(n):
      Append(R,random.randrange(101))
       # L.append(random.randrange(101))
    return R
#gets length of given list
def getLength(L):
    temp = L.head
    count = 0
    while temp is not None:
       temp = temp.next
       count += 1
    return count
############################################################
# copies list given into a new one
def copyList(N):
    # Prints list L's items in order using a loop
       clone = List()
       te = N.head
      #clone = List()
       while te is not None:
          Append(clone,te.item)
          te = te.next
       return clone
############################################################
# finds median
```

```
def findMiddle(t,halfIndex):
    iter = t.head
    for i in range(halfIndex):
        midElement = iter.item
        iter = iter.next
    return midElement
##########################################
def quickSort(L):
    # cheks that LL that is passed is large enough
    if getLength(L) > 1:
        # pivot is head of list
        pivot = L.head.item
        # lists that are going to store the two halves of the original list
        p1 = List()
        p2 = List()
        #stores the element after head into 't' because its already the pivot
        t = L.head.next
        while t != None:
            # if element is smaller than pivot its stored in p1 list
            if t.item < pivot:
                Append(p1, t.item)
             # if element is bigger than pivot its stored in p2 list
            else:
                Append(p2, t.item)
            t = t.next
        #recursively sorts the halves
        quickSort(p1)
```

```
        quickSort(p2)
        # inserts pivot into p1 if p1 is empty
        if IsEmpty(p1):
            Append(p1, pivot)
        #inserts pivot into p2s head if p1 is not empty
        else:
            Prepend(p2, pivot)
        #reconnects the halves of the list
        if IsEmpty(p1):
            L.head = p2.head
            L.tail = p2.tail
        else:
            p1.tail.next = p2.head
            L.head = p1.head
            L.tail = p2.tail
##########################################################
##########################################################
#length of list
numOfLLElements = 10
# half of list length
half = numOfLLElements //2
M = List()
M = randomListGen(numOfLLElements)
print('original list:')
Print(M)
print('')
C = List()
```

```
C = copyList(M)
D = List()
D = copyList(M)
E = List()
E = copyList(M)
print('list sorted though bubble sort:')
bubbleSort(C)
Print(C)
print('middle element of Linked List sorted by BubbleSort',findMiddle(C,half))
print('')
quickSort(D)
print('list sorted though Quick sort:')
Print(D)
print('middle element of Linked List sorted by QuickSort',findMiddle(D,half))
print('')
print('list sorted though merge sort:')
mergeSort(E)
Print(E)
print('middle element of Linked List sorted by merge Sort',findMiddle(E,half))
print('')
```

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 inappropiate assistance to any student in the class.

X