Lab 2 Report

CS 2302 Data Structures

University Of Texas at El Paso

Diego Quiñones
80582918

Source Code

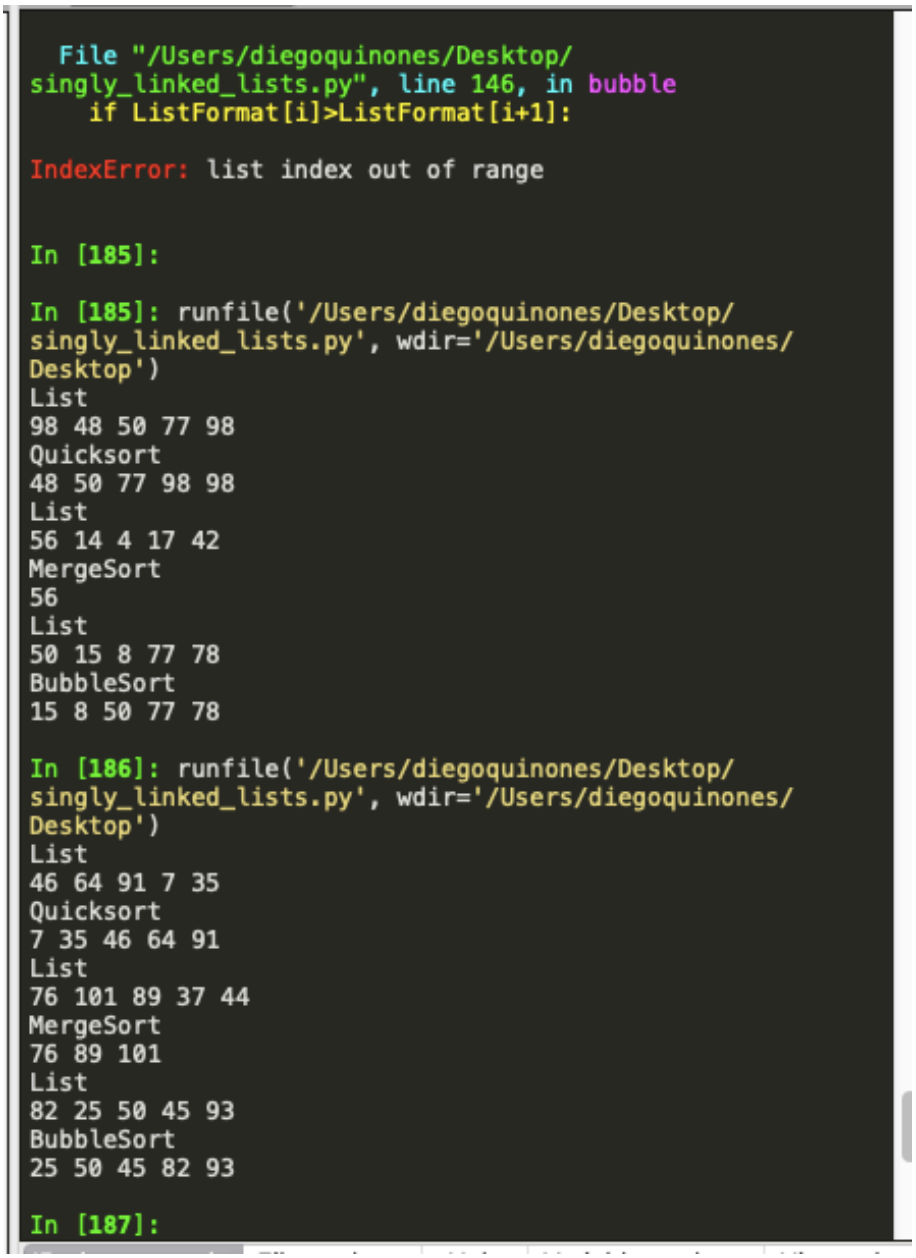
```
#Node Functions
import random
import time
class Node(object):
    # Constructor
    def __init__(self, item, next=None):
        self.item = item
        self.next = next

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 replace(L):
    temp=L.head
    Listing=[]
    while temp is not None:
        value=temp.item
        Listing.append(value)
        temp=temp.next
    return Listing

def AppendRandom(L):
    # Inserts x at end of list L
    if IsEmpty(L):
        L.head = Node(listFiller())
```

```
        L.tail = L.head
    else:
        L.tail.next = Node(listFiller())
        L.tail = L.tail.next

def Append(l1,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 GetLength(L):
    counter=0
    temp = L.head
    while temp is not None:
        counter=counter+1
        temp=temp.next
    return counter

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 listFiller():
    y=random.randint(1,101)
    return y

def mergeLists(l1, l2):
    temp = None
    if l1 is None:
        return l2
    if l2 is None:
        return l1
    if l1.item <= l2.item:
        temp = l1
        temp.next = mergeLists(l1.next, l2)
    else:
        temp = l2
        temp.next = mergeLists(l1, l2.next)
```

```
    return temp
#combines al of the lists
def mergeSort(head):
    if head is None or head.next is None:
        return head
    l1, l2 = divideLists(head)
    l1 = mergeSort(l1)
    l2 = mergeSort(l2)
    #makes a head based on the combination of the two lists
    head = mergeLists(l1, l2)
    return head
#divides lists between halfs
def divideLists(head):
    b = head
    a = head
    if b:
        b = b.next
    while b:
        b = b.next
        if b:
            b = b.next
            a = a.next
    mid = a.next
    a.next = None
    return mid,head

def quickSortPros(List1):
    more=[]
    same=[]
    less=[]
    if len(List1)>1:
        pivot=List1[0]
        for i in List1:
            if i < pivot:
                less.append(i)
            elif i == pivot:
                same.append(i)
            elif i > pivot:
                more.append(i)
        return quickSortPros(less)+same+quickSortPros(more)
    else:
        return List1

def quickSort(L):
    ListFormat=replace(L)
    Length=GetLength(L)
```

```
    ListFormat2=quickSortPros(ListFormat)
    L.head=L.tail
    L.head=None
    for i in range(Length):
        Append(L,ListFormat2[i])


def bubble(L):
    Length=GetLength(L)
    ListFormat=replace(L)
    for i in range(0,Length-1):
        for j in range(0,Length-1):
            if ListFormat[i]>ListFormat[i+1]:
                stor=ListFormat[i]
                ListFormat[i]=ListFormat[i+1]
                ListFormat[i+1]=stor
    L.head=L.tail
    L.head=None
    for i in range(Length):
        Append(L,ListFormat[i])

def Median(L):
    temp=L.head
    Length=round(GetLength(L)/2)
    for i in range(Length+1):
        value=temp.item
        temp=temp.next
    return value


print('List')

#create list
L = List()
for i in range(5):
    AppendRandom(L)


Print(L)
#sorting method
print('Quicksort')
#running time
start = time.time()
quickSort(L)
end = time.time()
#print running time
```

```
print(end - start)
Print(L)
print(Median(L))

#reset list
L.head=L.tail
L.head=None

for i in range(5):
    AppendRandom(L)
print('List')
Print(L)
#sorting method
print('MergeSort')
start = time.time()
mergeSort(L.head)
end = time.time()
print(end - start)
#print running time
Print(L)
print(Median(L))
#reset list
L.head=L.tail
L.head=None

for i in range(5):
    AppendRandom(L)
print('List')
Print(L)
#sorting method
print('BubbleSort')
start = time.time()
bubble(L)
end = time.time()
print(end - start)
#print running time
Print(L)
print(Median(L))
```


File "/Users/diegoquinones/Desktop/
singly_linked_lists.py", line 146, in bubble
if ListFormat[i]>ListFormat[i+1]:
IndexError: list index out of range
In [185]:
In [185]: runfile('/Users/diegoquinones/Desktop/
singly_linked_lists.py', wdir='/Users/diegoquinones/
Desktop')
List
98 48 50 77 98
Quicksort
48 50 77 98 98
List
56 14 4 17 42
MergeSort
56
List
50 15 8 77 78
BubbleSort
15 8 50 77 78
In [186]: runfile('/Users/diegoquinones/Desktop/
singly_linked_lists.py', wdir='/Users/diegoquinones/
Desktop')
List
46 64 91 7 35
Quicksort
7 35 46 64 91
List
76 101 89 37 44
MergeSort
76 89 101
List
82 25 50 45 93
BubbleSort
25 50 45 82 93
In [187]:

In Lab 2 I was presented with the task of using different types of sorting methods in List classes “Linked lists” . There were four sorting methods used in this:

Bubble sort: consists of comparing each value that is “next to the other” and switching them if the second value is larger than the first one.

Merge sort: divides the list in two multiple times, to the point where each element is by itself, and it start merging them by pairs, but know sorted.

Quick sort: uses the values at the most right or left, and uses it as a “pivot”, and moves values to each of the sides of the pivot based on if they are more or less than the value of the pivot.

The first step was creating the Node and list classes, so I used a code used in a previous activity. The I also had to create a code that appended random values on a List.

After that it was time for the sorting methods, so I used available online info as base to understand the logic of the code and how the sorting methods were supposed to be managed.

My biggest problem was managing the changes in the data from the linked list nodes because they work differently as regular lists, they aren’t as easy to access. Also the recursive calls sometimes confused me.

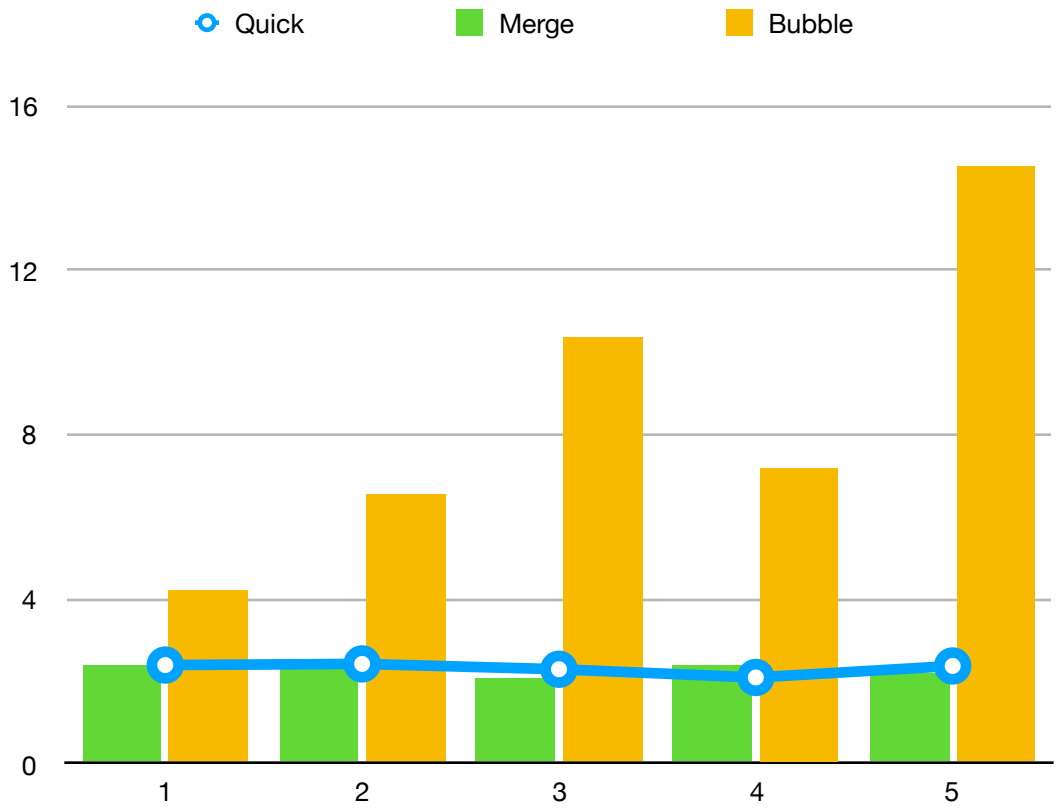
Running Times

Bubble Sort: O(n^2)

Merge Sort: O(nlogn)

Quick Sort: O(nlogn)

Running Times Chart



		1	2	3	4	5
	Quick	2.402	2.43	2.3	2.1	2.38
	Merge	2.408	2.408	2.1	2.408	2.2
	Bubble	4.234	6.568	10.3444	7.2149	14.50392