Lab 2 Adrian Monreal

In this lab Dr.Fuentes required us to use list and implement 3 different sort functions, bubble sort, merge sort, and quicksort. I had trouble with the syntax for my functions I understood the logic really well implementing the functions made sense however I kept receiving the same syntax errors.

Bubble Sort takes the unordered List given in the parameter and sorts using bubble sort compares each element to the one that follows if its greater they switch until it finds one bigger than it goes to the next element repeats until the list is sorted. I used a while loop and changed it to false every loop that was implemented and if it didn’t change back to true it would repeat the loop

The beginning of merge Sort this method splits the list in half it returns 2 lists to be inserted into the merge part of merge sort. each list is sorted it takes 2 list as parameters both are sorted so it inputs the first list, then it inputs the middle element then the second sorted list is applied. It made sense to me that the middle index would be the end point of the first list and the beginning point of the second list combining the 2 sorted list.

this Implementation of quick uses partion first and then it uses the quicksort to keep calling partition until the whole list is sorted. Since the the last element is the pivot the partition method will keep calling until it’s in order, because it will keep creating a new pivot until its entirely sorted. The logic I used was that if I do all the switching in one method and the second method will just call it until its an order was the simplest for me I didn’t need to have 2 recursive calls.

#Adrian Monreal  
#80570881  
  
import time  
  
  
# 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  
  
  
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()  
  
def RandomList(n):  
 pos = 0  
 RanList = List()  
  
 while pos < n:  
 curr = RanList.head  
 curr.item = random.randint(0, n)  
 # Append(RanList,random.randint(0, n))  
 curr = curr.next  
 pos += 1  
 return RanList  
  
  
#takes the unordered List given in the parameter and sorts using bubble sort  
#Bubble sort compares each element to the one that follows  
#if its greater they switch until it finds one bigger than it goes to the next element  
#repeats until the list is sorted  
def BubbleSort(L):  
 change = True  
 while change:  
 t=L.head  
 change = False  
 while t.next is not None:  
 if t.item > t.next.item:  
 temp = t.item  
 t.item = t.next.item  
 t.next.item = temp  
 change = True  
 t=t.next  
  
def length\_Of\_List(L):  
 if IsEmpty(L):  
 return 0  
 else:  
 L.head = L.head.next  
 return 1+ length\_Of\_List(L)  
#the Beginning of merge sort this method splits the list in half  
#it returns 2 list to be inserted into the merge part of merge sort  
# each list is sorted  
def split(L):  
 middle = length\_Of\_List(L)//2  
 firstHalf=List()  
 secondHalf=List()  
 curr = L.head  
 while i < middle:  
 firstHalf.head.item = curr.item  
 #Append(firstHalf,firstHalf.head.item)  
 i+1  
 while i< length\_Of\_List(L):  
 secondHalf.head.item = curr.item  
 #Append(secondHalf,secondHalf.head.item).  
 i+1  
 return firstHalf and secondHalf  
  
#takes 2 list as parameters both are sorted so it inputs the first list  
#then it inputs the middle element then the second sorted list is applied  
  
def merge(first,second):  
 CombinedList= list()  
  
 while first.head is not None:  
 CombinedList.head.item =first.head.item  
 #Append(CombinedList,first.head.item)  
 CombinedList.head = CombinedList.head.next  
 first.head = first.head.next  
 middleElement = CombinedList.head.item  
 while second.head is not None:  
 CombinedList.head.item =second.head.item  
 #Append(CombinedList,second.head.item)  
 CombinedList.head = CombinedList.head.next  
 second.head = second.head.next  
 return [CombinedList, middleElement]  
  
#this Implementation of quick uses partion first and then it uses the quicksort  
# to keep calling partition until the whole list is sorted  
#Since the the last element is the pivot the partition method will keep calling until its in order  
#because it will keep creating a new pivot until its entirely sorted  
  
def partion(L):  
 PL = L #PartionedList  
 curr = PL.head  
 newHead = PL.head  
 pivot = PL.tail  
 newTail = PL.tail  
 prev = curr  
 while curr != pivot :  
 if newHead.item > pivot.item:  
 TempTail = curr  
 curr = curr.next  
 newTail.next = TempTail  
 if curr.item > pivot.item:  
 TempTail = curr  
 curr = curr.next  
 newTail.next = TempTail  
 prev.next = curr  
 else:  
 curr = curr.next  
 return [pivot, PL]  
  
def inOrder(L):  
 curr = L.head  
 while curr is not None:  
 if curr.item > curr.next.item:  
 return False  
 curr = curr.next  
 return True  
  
  
def quicksort(PL): #takes partioned List as a parameter  
 if not inOrder(PL):  
 newList = partion(PL)  
 return quicksort(newList)  
 return PL  
  
  
L = List()  
print(IsEmpty(L))  
for i in range(5):  
 Append(L, i)  
  
print(" Bubble Sort")  
start = time.time()  
print(BubbleSort(L))  
end = time.time()  
print(end - start)  
  
print("------------------")  
start = time.time()  
print("Merge Sort")  
[left,right] = split(L)  
print(merge(left,right))  
end = time.time()  
print(end - start)  
  
print("------------------")  
  
start = time.time()  
print("Quick Sort")  
PartList = partion(L)  
print(quicksort(PartList))  
end = time.time()  
print(end - start)

I Adrian Monreal 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.