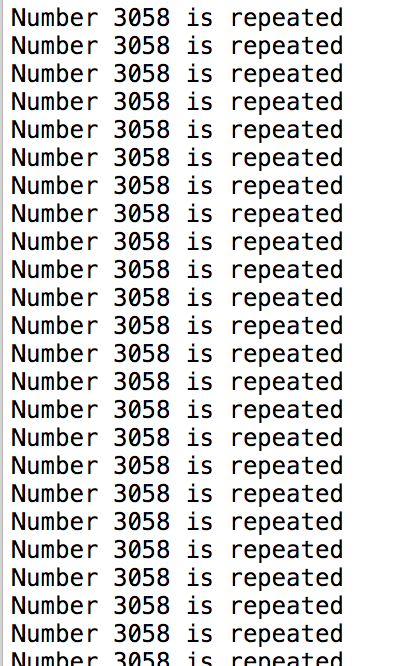
During this lab the objective was to combine 2 .txt documents with integers, considered employee ids, into a single liked list. The employee ids where obtained from the merging of 2 companies, which decided to check for any repetitions in the id lists. I tried to solve this problem by creating the single linked list and by using the .readLines() method for each file. By creating a linkedList class, I was able to create methods to edit and manipulate my created list as a whole. It was required for me to implement 4 different solutions, which I will describe next.

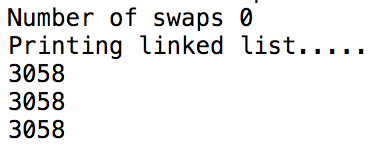
For solution one, I implemented two while loops in the method nestedLoops() to check every element in the list against the other remaining elements. The 2 nested while loops caused this method to be O(n2) where n is the length of my linked list. This method printed if the number was repeated, but printed a statement the number of times it was found in the list. For example, if the number 2 was found 3 times, the program printed the line “Number 2 is repeated” three times. This problem was solved in solution 4.

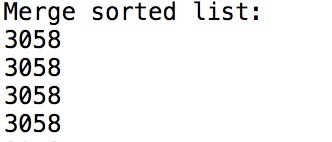
In solution two, I implemented the bubbleSort() method to my list. This method was created as part of the linkedList class.I also printed the number of swaps done by the method to create a visual representation of the number of swaps the method does. In my method I checked if the list was empty and if the list contained only one element, if true the method had a O(1); if the list was not empty and contained more than one element, I used while loops to check every element in the list against the remaining elements. The two nested while loops increased the time complexity of my method to O(n2) where n is the length of my linked list.

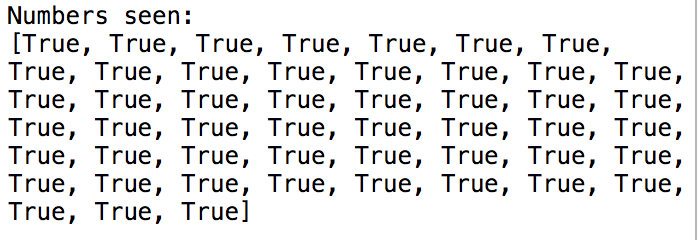
For the third solution, I implemented mergeSort() to my list. To implement merge sort I used three methods, mergeSort(), merge() and middle(). In the mergeSort() method I called the middle() method first to cut the list in half, then I called the mergeSort() method recursively for the two lists to be cut until the left and right lists where of size 0. After the lists where cut, then I merged them in order with the method merge(). The running time of this algorithm is O(n log n) since the lists are being cut in half every time and then ordered and merged together. The only issue I had with this method was that it returned a Node type object instead of a list. After calling the method, I had to use a while loop to check if the elements where ordered correctly by the method.

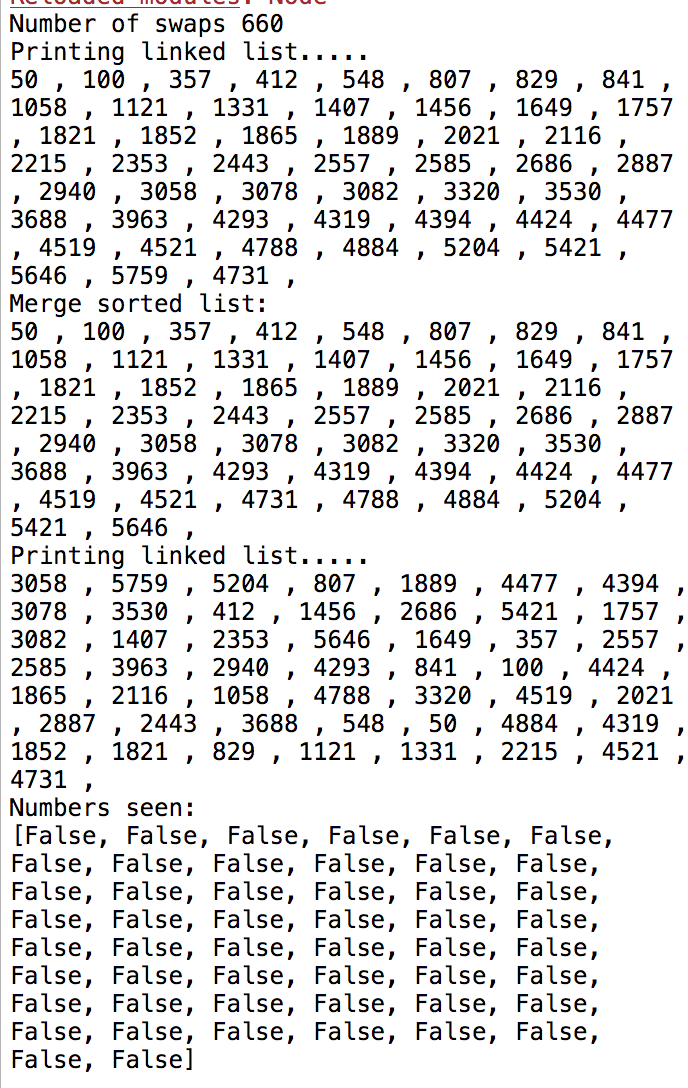
For the final solution I used the methods searchRepeat() and isRepeated(). The method isRepeated used a copy of the unsorted initial list ids\_list as a parameter. In this method I created an array with the elements that where repeated in the list called repeatArray[]. I used an if statement to check if curr and nodeCompare had the same information and also checked if the item wasn’t already in the repeatArray[], if not then the repeated number would be appended into the list. Also, I created an array named listArray[] where I copied the information of my list and placed it in an array. Basically, I copied the list into an array. After this, I called the searchRepeat() method which returned a Boolean value whether the number was found in the repeatArray[] or not. I had trouble understand the instructions of the lab, I wasn’t sure if I needed to give a Boolean value to every digit in the list whether it was repeated or not, but that’s how I interpreted the instructions. Also, I created the array repeatArray[] to save any repeats found in the list. The running time of this algorithm is dependent on the size of the list, and placing copies into the listArray[] while also traversing the list to create the seenArray[]. Each array creation is O(n).

 For my first test case I ran a file 50 identical numbers. This stamen was repeated 50 times because of my solution 1.

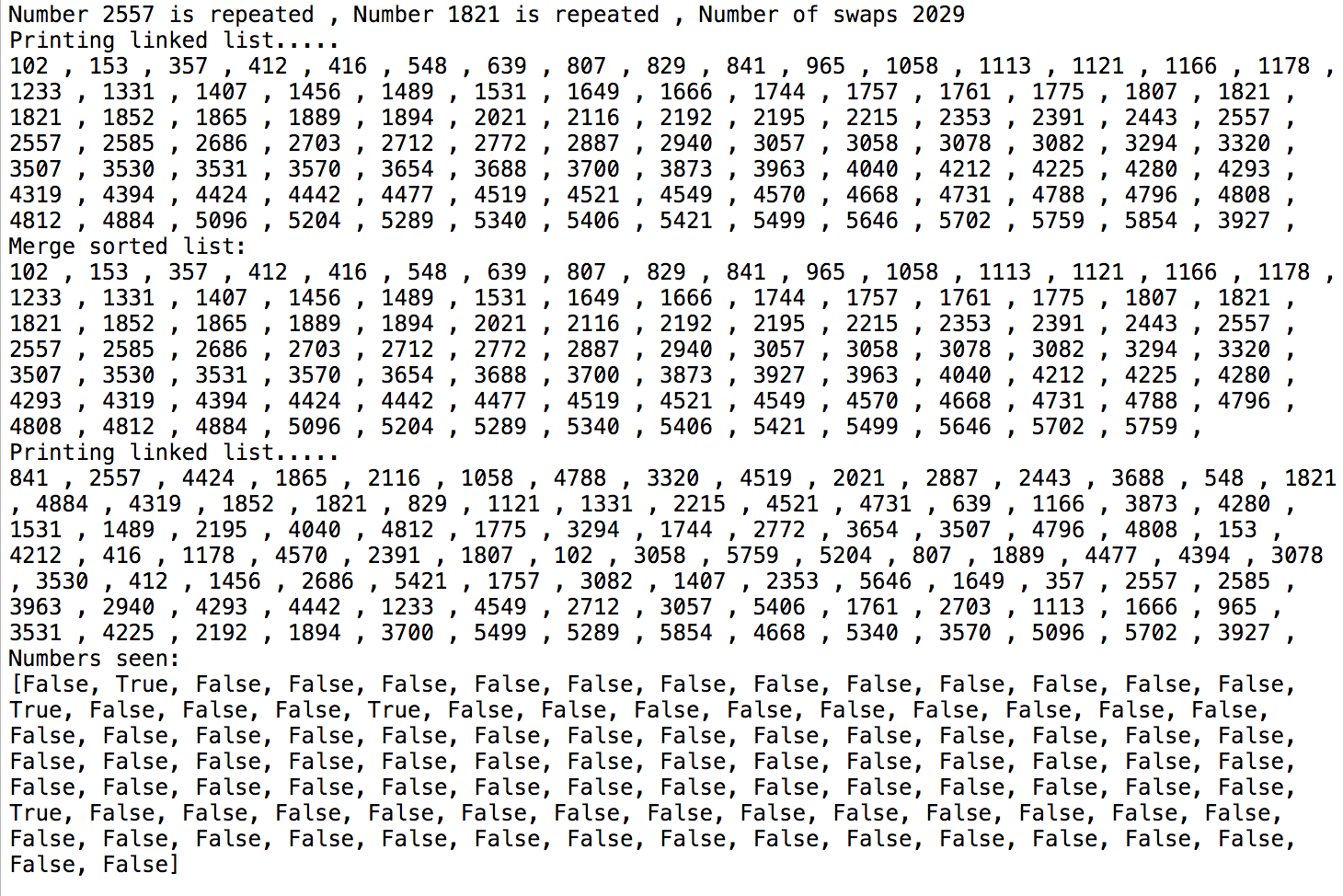
My bubbleSort() method didn’t do any swaps. Since the numbers where all the same.

The merge sorted list printed the number 50 times.

My seenArray[] printed the following since it had seen all the numbers in the listArray[] before.

 For my second test case I used a list with 50 unique numbers. The program printed the following. After the first Printing linked list…, the method prints the sorted list using bubbleSort()﻿. Then prints the merge sorted list. After this, it prints the copy of the original unsorted list, and the finally it prints the seen [].

For my third test case I used a random list with 100 numbers. The list has repeated numbers. It printed the following:



The big Oh difference in running time of bubble Sort against merge Sort in my program was represented by the number of swaps in bubble sort against the number of cuts in merge sort.

For this lab I learned how to implement sorting algorithms in a linked list. I had never implemented them on a list before, only arrays, so this lab was a challenge for me. Also, I noticed that when you input a huge file, like the one originally provided, the bubble sort algorithm takes minutes to finish. I kind of expected this because of the concepts that have been thought before about the running times of the sorting algorithms. I also had trouble with the instructions for solution 4, I’m not sure I understood them correctly like I mentioned before.

Source Code:

﻿"""

Created on Sun Oct 14 20:37:27 2018

@author: Diana Ramirez 88604827

Professor: Diego Aguire

Class: CS 2302 TR 1:30PM

Lab 2A

"""

from Node import Node

class linkedList(object):

def \_\_init\_\_(self, head = None):

self.head = head

def addNew(self,int):

newNode = Node(int)

if self.head == None:

self.head = newNode

else:

temp = self.head

while temp.next != None:

temp = temp.next

temp.next = newNode

def lengthlist(self):

temp = self.head

length = 0

while(temp.next != None):

length+=1

temp = temp.next

return length

def printList(self):

print("Printing linked list.....")

if(self.head is None):

return;

temp = self.head

while(temp!= None):

print(temp.item,",",end =" ")

temp = temp.next

print()

def copy(self):

final = linkedList()

curr = self.head

while curr != None:

final.addNew(curr.item)

curr = curr.next

return final

#####################solution 1, nested loops ####################

def nestedLoops(self):

nodeCompare = self.head

while(nodeCompare != None):

curr = nodeCompare.next

while(curr != None):

if(nodeCompare.item == curr.item):

print("Number",curr.item,"is repeated",",",end =" ")

curr = curr.next

nodeCompare = nodeCompare.next

#############solution 2 : bubbleSort method for the list##############

def bubbleSort(self):

swaps=0

if(self.head == None):

print("No nodes to sort, list is empty")

elif self.head.next == None:

return self.head

else:

temp = self.head

while temp.next != None:

curr = temp.next

while curr.next != None:

if temp.item > curr.item:

##where the swapping of data occurs

dummy = temp.item

temp.item = curr.item

curr.item = dummy

swaps+=1

curr = curr.next

temp = temp.next

print("Number of swaps", swaps)

#########################solution 3: merge Sort##########################

##actual mergeSort method that calls the middle() and mergeLists() methods recursively

##this is outside of the linked list class

def mergeSort(head):

if (head == None or head.next == None):

return head

left, right = middle(head)

left = mergeSort(left)

right = mergeSort(right)

head = merge(left,right)

return head

##method to unite both halfs of the list

def merge(left,right):

temp = None

if left == None:

return right

if right == None:

return left

if left.item <= right.item:

temp = left

temp.next = merge(left.next,right)

else:

temp = right

temp.next = merge(left,right.next)

return temp

##method used to cut the lists into 2

def middle(head):

slow = head ##decides middle of list

fast = head ##goes all the way to the end of list

if fast:

fast = fast.next ##used to incremeten fast twice as much as slow

while fast:

fast = fast.next

if fast:

fast = fast.next

slow = slow.next

mid = slow.next

slow.next = None

return head, mid

###############################solution 4#####################################

def searchRepeat(listArray,repeatArray):

##seen is a boolean array that stores if values are repeated in list

seen = []

##searching in list array

for x in range (len(listArray)):

y = listArray[x]

##if number in listArray is found in repeatArray add True to seen

if y in repeatArray:

seen.append(True)

##number not found in repeatArray so add false

else:

seen.append(False)

print("Numbers seen: ")

print(seen)

return seen

def isRepeated(list):

repeatArray = []

listArray = []

curr = list

nodeCompare = list

while(nodeCompare != None):

curr = nodeCompare.next

while(curr != None):

##if statement used to create repeatArray

if(nodeCompare.item == curr.item and curr.item not in listArray):

##creating an array of repeated numbers in the list

repeatArray.append(curr.item)

curr = curr.next

##storing the elements of the list in an array

listArray.append(nodeCompare.item)

nodeCompare = nodeCompare.next

##calling method to create boolean array

searchRepeat(listArray,repeatArray)

##############################################################################

##opening the files

activision = open('activision.txt')

vivendi = open('vivendi.txt')

##initializing the linkedList

ids\_list = linkedList()

##separates and reads lines of file

linesActi = activision.readlines()

linesVive = vivendi.readlines()

##adds numbers in Activision file to the ids list

for ln in linesActi:

ln = int(ln)

##using the LinkedList class

ids\_list.addNew(ln)

##adds numbers in Vivendi file to the ids list

for ln2 in linesVive:

ln2 = int(ln2)

##using the LinkedList class

ids\_list.addNew(ln2)

##making a copy of the unsorted list

copy = ids\_list.copy()

#####################solution 1######################

ids\_list.nestedLoops()

######################solution 2#######################

ids\_list.bubbleSort()

ids\_list.printList()

######################solution 3########################

print("Merge sorted list:")

##returns nodes not a list, saved in node ids

ids = mergeSort(ids\_list.head)

##printing the sorted nodes

dummy = ids

while(dummy.next != None):

print(dummy.item,",",end =" ")

dummy = dummy.next

print()

######################solution 4#########################

copy.printList()

###used the unsorted copy of the original list

isRepeated(copy.head)

“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 inappropriate assistance to any student in the class.”

