**Lab 2 Report**

The main purpose of this lab was to use different implementations for example, merge sort, bubble sort. I was also going to test and determine if there were any duplicates in the list by comparing each of the items. One of the problems that I encountered where the implementations with bubble sort and merge sort because at first, they were not running effectively and then had to go with the peer leader for clarification and the TA in a class to still ask more questions. The file was not being read properly and could not read line by line as it was asked. In order to test my file, I created a file where there was a lot of different numbers that were repeating. However, this was a small file in order to see if the program is first working effectively and implementing the code correctly. After testing it with the small file I used activision.txt a bigger file that allowed me to see all the repeated numbers by comparing them once merge sort and bubble sort was apply to the list of small and big numbers. What I learned is that merge sort is a much quicker sorter than bubble sort since it takes around 10.1 sec or n^2 more than merge sort with a time complexity of O(nlogn) taking less than a second to run the program. I also learned how to implement these methods but most importantly I know that with a bigger list I should always use merge sort in order to save memory and space. This was definitely a challenging lab and could still need more practice. (Since I had a really hard time).

# CS2302 Data Structures  
# Programmed by Luis Garcia.  
# Last modified November 28, 2018.  
# Instructor Diego Aguirre.  
# Implementation of  
# lab 2  
  
  
class Node(object):  
 def \_\_init\_\_(self, data, next):  
 self.data = data  
 self.next = next  
  
  
class LinkedList(object):  
 def \_\_init\_\_(self, head=None):  
 self.head = head  
 self.size = 0  
  
 # Inserts new node into the linked list  
 def insert(self, data):  
 if self.head is None:  
 self.head = Node(data, None)  
 self.size += 1  
 else:  
 temp = self.head  
 new\_node = Node(data, None)  
 while temp.next is not None:  
 temp = temp.next  
 temp.next = new\_node  
 self.size += 1  
  
 def getSize(self):  
 return self.size  
  
 def getHead(self):  
 return self.head  
  
 # This method computes the linked list and checks if there are any duplicates already present  
 def solution1(self):  
 currNOde = self.head  
 # nested loop to compare every element  
 while currNOde is not None:  
 nextNode = currNOde.next  
 while nextNode is not None:  
 if currNOde.data == nextNode.data:  
 print("Duplicate found:", currNOde.data)  
 nextNode = nextNode.next  
 currNOde = currNOde.next  
  
 # This method completes bubble sort on the linked list  
 def solution2(self):  
 yes\_no = True  
 while yes\_no is True:  
 yes\_no = False  
 temp = self.head  
 while temp.next is not None:  
 if temp.data > temp.next.data:  
 holds\_place = Node(temp.data, None)  
 temp.data = temp.next.data  
 temp.next.data = holds\_place.data  
 yes\_no = True  
 temp = temp.next  
  
 # This Method creates a new array of boolean variables and updates to true if node has already been seen or inserted  
 def solution4(self):  
 temp = self.head  
 size = self.getSize() + 1  
 boolean = [False] \* size  
 # used to keep track of previously visited nodes  
 seen\_numbers = []  
 duplicates\_counter = 0  
 while temp is not None:  
 if temp.data in seen\_numbers:  
 boolean[temp.data] = True  
 duplicates\_counter += 1  
 else:  
 seen\_numbers.append(temp.data)  
 temp = temp.next  
 temp = self.head  
 # Printing of the array of seen nodes  
 while temp is not None:  
 print("There are duplicates of ", temp.data, ": ", boolean[temp.data])  
 temp = temp.next  
 print("The total number of found repeats is:", duplicates\_counter)  
  
  
# Prints Linked List  
def printList(head):   
  
 if head is None:  
 print("Linked List is empty")  
 else:  
 temp = head  
 num\_elements = 0  
 while temp is not None:  
 print(temp.data)  
 num\_elements += 1  
 temp = temp.next  
 print("Total number of elements is:", num\_elements)  
  
  
# This is used for merge sort to split the list into two parts and continue to do so recursively  
def separateList(head):  
 if head is None or head.next is None:  
 return head, None  
 else:  
 mid\_node = head  
 end\_node = head.next  
 while end\_node is not None:  
 end\_node = end\_node.next  
 if end\_node is not None:  
 mid\_node = mid\_node.next  
 end\_node = end\_node.next  
  
 # Beginning of the first list  
 first\_half = head  
 # Sves where second head of list s  
 second\_half = mid\_node.next  
 # Used to end and start both new lists  
 mid\_node.next = None  
 return first\_half, second\_half  
  
  
# This method is used to merge the two lists that have been split but now sorted  
  
def mergeSort(first\_half, second\_half):  
 new\_head = Node(None, None)  
 temp = new\_head  
 while first\_half is not None and second\_half is not None:  
 if first\_half.data <= second\_half.data: # If  
 temp.next = first\_half  
 first\_half = first\_half.next  
 else:  
 temp.next = second\_half  
 second\_half = second\_half.next  
  
 temp = temp.next  
 if first\_half is None:  
 temp.next = second\_half  
 elif second\_half is None:  
 temp.next = first\_half  
 return new\_head.next  
  
  
# Main Merge Sort operation calling for spilt and merge of the linked list  
  
def solution3(head):  
 if head is None or head.next is None:  
 return head  
 else:  
 # List split in half to sort  
 first\_half, second\_half = separateList(head)  
 first\_half = solution3(  
 first\_half)  
 second\_half = solution3(second\_half)  
 # Sorted list is merged together to form the final complete list  
 return mergeSort(first\_half, second\_half)  
  
  
def main():  
 file\_activision = open("words.txt", "r")  
 file\_vivendi = open("nums.txt", "r")  
 bubble\_List = LinkedList()  
 merge\_List = LinkedList()  
 solution4\_List = LinkedList()  
  
 for line in file\_activision:  
 number\_id = int(line.strip())  
 bubble\_List.insert(number\_id)  
 merge\_List.insert(number\_id)  
 solution4\_List.insert(number\_id)  
  
 for line in file\_vivendi:  
 number\_id = int(line.strip())  
 bubble\_List.insert(number\_id)  
 merge\_List.insert(number\_id)  
 solution4\_List.insert(number\_id)  
  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print("Test for Solution #1:")  
 print("The following duplicates were found:")  
 bubble\_List.solution1()  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print("Test for Solution #2:")  
 print("The linked list will now be sorted using bubble sort.")  
 bubble\_List.solution2()  
 bubble\_head = bubble\_List.getHead()  
 printList(bubble\_head)  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print("Test for Solution #3:")  
 print("The linked list will now be sorted using merge sort.")  
 head = merge\_List.getHead()  
 sorted\_LL = solution3(head)  
 printList(sorted\_LL)  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print("Test for Solution #4")  
 solution4\_List.solution4()  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
  
  
main()

“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 class.”