For this lab I separated the code in three parts: the Node object, the EmployeeID object, and the main method. Inside the Node object I defined the different parts of the node, which were ‘next,’ ‘prev,’ and ‘data.’

Inside the EmployeeID object method I applied the node with different methods such as length (which calculated the length of the list), insert (which added new nodes to the beginning list), append (which added new nodes to the end of the list), printList (which printed the list in ascending order), printItem (which prints a single node), toInt (which converts any non-integer data into integer), and search (which searched a specific node in the list).

Inside the main method I implemented the methods of the EmployeeID object and expanded them. The methods inside were read\_lines (reads lines in a file and creates a list), compareAll (compares all items in list using the search method), bubbleSort, mergeSort, splitList, and mergeLists (combines both lists into one).

In order to make things work, I had to first create two lists out of the two files provided and then combine them. The main problem (and biggest problem) that I ran into was the fact that my file wouldn’t compile because I couldn’t figure out how to call the data function out of the Node object through the main method.

I literally tried for days until my brain was fried. Which is also why I couldn’t test it.

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.

*class* Node:  
 *def \_\_init\_\_*(*self*,data):  
 *self*.next = None  
 *self*.prev = None  
 *self*.data = data  
  
*class* EmployeeID:  
  
 *def \_\_init\_\_*(*self*):  
 *self*.head = None  
  
 *def \_\_len\_\_*(*self*):  
 *return self*.length  
  
 *# Add new item to the beginning of the list  
 def* insert(*self*, new\_data):  
 new\_id = Node(data = new\_data)  
 new\_id.next = *self*.head  
 new\_id.prev = None  
  
 *if self*.head *is not* None:  
 *self*.head.prev = new\_id  
  
 *self*.head = new\_id  
  
 *# Add new item to the end of the list  
 def* append(*self*, new\_data):  
 new\_id = Node(data = new\_data)  
 last = *self*.head  
 new\_id.next = None  
  
 *if self*.head *is* None:  
 *self*.head = new\_id  
 *return  
  
 while* (last.next *is not* None):  
 last = last.next  
  
 last.next = new\_id  
 new\_id.prev = last  
  
 *# Print list in ascending order  
 def* printList(*self*, node):  
 *while* node *is not* None:  
 *# Print on separate lines  
 print*(node.data)  
 *#Print on same line  
 # print(node.data, end=" ")* last = node  
 node = node.next  
  
 *def* printItem(*self*, node):  
 *print*(node.data)  
  
 *def* toInt(*self*, node):  
 node.data = int(node.data)  
 *return* node.data  
  
 *# Search for key  
 def* search(*self*, key):  
 curr = *self*.head  
 isFound = False  
 counter = 0  
 *while* curr != None:  
 *if* curr.data == key:  
 isFound = True  
 counter = counter +1  
 curr = curr.next  
 *#if counter > 1:  
 #print("ID #", key, " found ", counter, " times")  
 return* isFound  
  
*# Read each line and create linked list  
def* read\_lines(file\_name, h):  
 list = h  
 *with* open(file\_name, 'r') *as* textFile:  
 *for* line *in* textFile:  
 line = line.rstrip("\n")  
 list.append(line)  
 *return* h  
  
*# Compare all items using search()  
def* compareAll(llist):  
 *while* llist.head != None:  
 key = llist.head.data  
 llist.search(key)  
 llist.head = llist.head.next  
  
*def* bubbleSort(llist):  
 h = llist  
 new\_h = h  
 curr = h  
 a = h  
 temp = h  
 *while* h.head.next != None:  
 curr.head = h.head.next  
 prev = h  
 a.head = curr.head.next  
 *while* a != None:  
 *if* (a.head.data < curr.head.data):  
 temp.head = a.head.next  
 a.head.next = prev.head.next  
 prev.head.next = curr.head.next  
 curr.head.next = temp.head  
 prev = a  
 a = temp  
 *else*:  
 a.head = a.head.next  
 curr.head = curr.head.next  
 prev.head = prev.head.next  
 h.head = h.head.next  
 new\_h.head = new\_h.head.next  
  
*def* mergeSort(llist):  
 *if* llist *is* None *or* llist.head.next *is* None:  
 *return* llist  
 left\_h = splitList(llist)  
 right\_h = splitList(llist)  
  
 left = mergeSort(left\_h)  
 right = mergeSort(right\_h)  
 *return* mergeLists(left, right)  
  
*def* splitList(llist):  
 *if* llist *is* None *or* llist.head.next *is* None:  
 left\_h = llist  
 right\_h = None  
 *return* left\_h, right\_h  
 *else*:  
 mid = llist  
 front = llist.head.next  
  
 *while* front *is not* None:  
 front = front.head.next  
 *if* front *is not* None:  
 front = front.head.next  
 mid = mid.head.next  
 left\_h = llist  
 right\_h = mid.head.next  
 mid.head.next = None  
 *return* left\_h, right\_h  
  
*def* mergeLists(left, right):  
 h = None  
 curr = h  
  
 *while* left *and* right:  
 *if* left.head.data < right.head.data:  
 curr.head.next = left  
 left = left.head.next  
 *else*:  
 curr.head.next = right  
 right = right.head.next  
 curr = curr.head.next  
  
 *if* left *is* None:  
 curr.head.next = right  
 *elif* right *is* None:  
 curr.head.next = left  
 *return* h.head.next  
  
*# Create activision list*activision = EmployeeID()  
activision = read\_lines('activision.txt', activision)  
  
*# Create vivendi list*vivendi = EmployeeID()  
vivendi = read\_lines('vivendi.txt', vivendi)  
  
*# Append both lists*full\_list = activision  
full\_list.append(vivendi)  
compareAll(full\_list)  
full\_list.printList(full\_list.head)  
bubbleSort(full\_list)  
  
*### TRAVERSAL*n = EmployeeID()  
n.head = full\_list.head.next  
n.printItem(n.head)  
n.head = n.head.prev  
n.printItem(n.head)