CISC 160
Lab 02

Background
Linked Lists are essential and elemental concrete data structures across languages. They also lead into later concepts such as trees and graphs. We need to be able to understand and manipulate linked lists and nodes in order to be able to choose the appropriate data structure for a given algorithm as well as be able to design and manipulate trees and graphs.

Sorting
One of the benefits of an array implementation is that sorting is a relatively easy task: arrays have slots where data can be stored so swapping the data is as simple as declaring a holder variable to do the swap. In Python, this is even easier as the following syntax can be used:

```
array[i], array[j] = array[j], array[i]
```

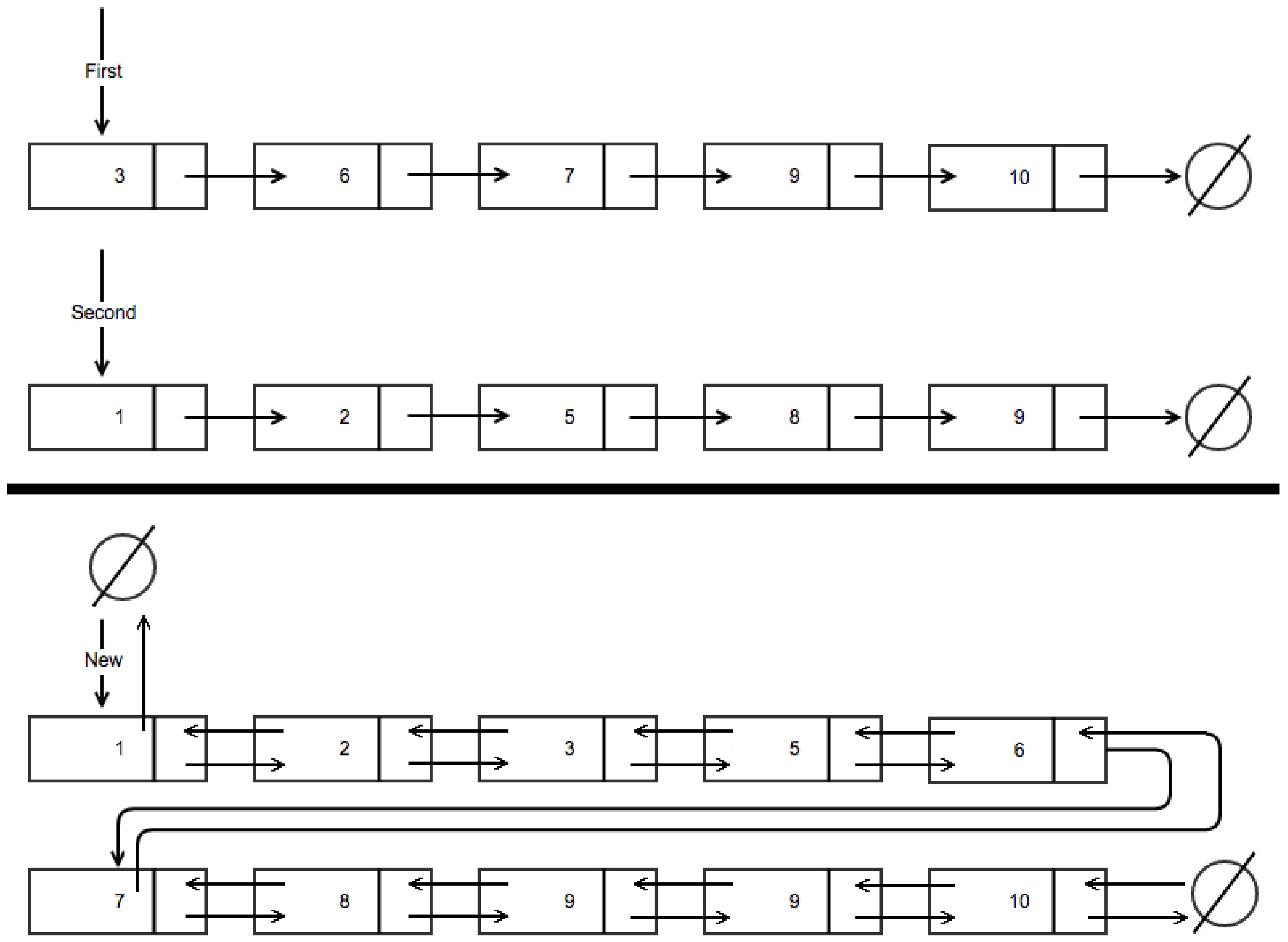
However, if you want to be sure that a linked list swapped, this involves moving pointers.

1. Download the files Node.py and sorted.py. Use (and do not copy or alter) the Node object defined in Node.py to implement the `insert_sorted` function declared in sorted.py. As mentioned in the associated comment, this function should take a Node which is considered the head of an already sorted singly linked list of nodes and an element which is to be inserted into the list. This insertion should be done in a way such that the list ***stays in order***. When you are done, you should return the Node at the head of the list.

   NOTE 1: You may assume that elements in the linked list and elements being added to the linked list are compatible with respect to equivalence and comparison operators.

   NOTE 2: Please be sure to examine and use the publicly available methods of the Node object, not the hidden attributes.

Combination
Sometimes lists need to be combined. If we have lists that are being combined and we know that the lists are already ordered, then we should probably combine them so that they stay in order. Below is a diagram of how this would look when combining two singly linked lists into a doubly linked list.



2. Download the files Node.py, Double_Node.py, and combined.py. Use (and do not copy or alter) the DNode object defined in Double_Node.py to implement the `combine_sorted` function declared in combined.py. As mentioned in the associated comment, this function should take two different Nodes which are considered the heads of two already sorted singly-linked lists. This function then combines them into a new, doubly-linked list which ***stays in order***. The original singly-linked lists should ***remain unaltered by the function and in their original state.*** When you are done, you should return the DNode at the head of the list.

   NOTE 1: You may assume that elements in the two linked list are compatible with respect to equivalence and comparison operators.

   NOTE 2: Please be sure to examine and use the publicly available methods of the Node and DNode objects, not the hidden attributes.

Lab Requirements
Download all source code files on the Canvas page and alter sorted.py and combined.py as described above. You may add any additional helper functions as needed. Helper functions should be named with an underscore at the front as they are not designed to be publicly accessible If you add a helper function, be sure to comment it in the same style as the other, included functions.

Additionally, the "testing structure" of `if __name__ == '__main__'` has been included at the bottom of each file. You should use that section for your test code. I may look at it for understanding of how you are approaching testing but you will not be explicitly graded on code included there.

Remember: These requirements may not be all encompassing. Use your brain, your knowledge of the system, and any descriptions within the code as sanity checks and reminders to make a complete system.

Submission: Submit your altered sorted.py and combined.py code files.