**Problem 1**

In class, we’ve studied Singly-Linked Lists which are made of nodes that point at subsequent nodes.

One of the biggest annoyances with Linked Lists is the difficulty of going backwards through a list

(such as getting the previous node or traversing the list backwards).

An intuitive solution to this inefficiency is the doubly-linked list, which adds pointers to previous

nodes. Doubly-Linked Lists are not very different from Singly-Linked Lists, but are far more

common because they are easier to use.

In this problem, you are to implement a Doubly-Linked List from scratch (you may use the Singly-

Linked List code from class). You will have to create 2 classes (Node and DoublyLinkedList).

The Node class will have the following methods:

\_ \_\_init\_\_(value, prev, next)

\_ get\_prev()

\_ get\_next()

\_ get\_value()

\_ set\_prev(node)

\_ set\_next(node)

\_ set\_value(val)

The DoublyLinkedList class must have the following methods:

\_ \_\_init\_\_(): Note that there are no extra parameters here

\_ add\_to\_end(val): adds element as last

\_ add\_to\_front(val): adds element to first

\_ delete(val): deletes first occurrence of val

\_ reverse(): reverses the list

\_ compare(lst): check if regular Python list has the same values in the same order as the DLL

\_ find(val): return the index(as it would be in a list) of the first occurrence of val

**Problem 2**

Write a function that will sort a given list using merge sort. You must implement and use the merge

sort algorithm (but may be recursive or iterative). The function will take a list as an input and

return a sorted version of the list (you may assume it will be a list of integers).

The method signature must be merge\_sort(lst).