CSC 225 SUMMER 2015
ALGORITHMS AND DATA STRUCTURES I
PROGRAMMING ASSIGNMENT 2
UNIVERSITY OF VICTORIA

1 Programming Assignment

The programming assignment is to implement the merge sort algorithm with the following input and output specification:

Input: A linked list L containing n non-negative integers.
Output: A linked list containing the elements of L in sorted (ascending) order.

Pseudocode for merge sort is given below:

1: **procedure** MERGESORT(L)
2: Split L into two lists L_1, L_2 of size (approximately) $n/2$
3: $S_1 \leftarrow$ MERGESORT(L_1)
4: $S_2 \leftarrow$ MERGESORT(L_2)
5: Merge the two sorted lists S_1 and S_2 together to a single sorted sequence S
6: **return** S
7: **end procedure**

You may not use arrays of any kind, or any of the collection types provided by the Java standard library (such as `ArrayList`, `LinkedList` or `Vector`).

To receive full marks on this assignment, your code is not permitted to contain any `for`, `while` or `do-while` loops, or any other iterative looping structures[1]. All looping behavior must be implemented with recursion. A completely correct solution, using iterative loops for both the split phase (line 2 of the pseudocode above) and the merge phase (line 5 of the pseudocode above) will receive at most 70% of the available marks (see the 'Evaluation Criteria' section below). If you prefer iterative code to recursive code, you may want to implement the algorithm iteratively first, then refine your solution to use recursion only.

Beyond the requirement that the implemented algorithm must be merge sort, there are no restrictions on how the split and merge operations behave. For example, you may find that splitting the list by taking alternating elements (instead of dividing the list in half at the middle) is easier to implement with recursion.

A Java template has been provided containing an empty function `MergeSort`, which takes the head of a singly-linked list of integers as its only argument. Your task is to write the body of the `MergeSort` function. You must use the provided Java template as the basis of your submission, and put your implmentation inside the `MergeSort` function in the template. You may not change the name, return type or parameters of the `MergeSort` function. You may add additional functions as needed. A class called `ListNode` has been provided in the template to represent list nodes. Both the parameter value and return value of `MergeSort` will have type `ListNode`. You are not permitted to change any aspect of the `ListNode` class (including adding, removing, or renaming its contents). However, you are free to create a subclass of `ListNode` if

1. Including elements of the Java library which emulate loops.

you want to extend its functionality. Since you are are only permitted to submit one file, any extra classes must be contained in the `MergeSort.java` file.

The `main` function in the template contains code to help you test your implementation by entering test data or reading it from a file. You may modify the `main` function, but only the contents of the `MergeSort` function (and any functions you have added) will be marked, since the `main` function will be deleted before marking begins. Please read through the comments in the template file before starting.

2 Test Datasets

Several files of test data have been uploaded to conneX. The Assignment 1 test data can be used as well.

The uploaded files may not cover all possible cases, so you should test your implementation on other inputs (particularly special cases, such as lists of size 0 or 1). One option is to write a short program to generate lists of integers for testing.

3 Evaluation Criteria

The programming assignment will be marked out of 30, based on a combination of automated testing (using large test arrays similar to the ones posted on conneX) and human inspection.

Score (/30)	**Description**
$0-5$	Submission does not compile or does not conform to the provided template.
$6-10$	The implementation uses arrays or data structures from the Java standard library.
$11-15$	The implemented algorithm is not merge sort or is substantially inaccurate on the tested inputs.
$16-21$	The implementation uses `for`, `while` or `do-while` loops for both the split and merge phases, but is otherwise correct and has a $\Theta(n \log n)$ running time.
$22-26$	The implemented algorithm is correct, has a $\Theta(n \log n)$ running time, and uses iterative loops for only one of the split and merge phases (and recursion for the other).
$27-30$	The implemented algorithm is correct, uses recursion for all looping behavior (and contains zero iterative loops), and has a $\Theta(n \log n)$ running time.

To be properly tested, every submission must compile correctly as submitted, and must be based on the provided template. **If your submission does not compile for any reason (even trivial mistakes like typos), or was not based on the template, it will receive at most 5 out of 30**. The best way to make sure your submission is correct is to download it from conneX after submitting and test it. You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. conneX will allow you to change your submission before the due date if you notice a mistake. After submitting your assignment, conneX will automatically send you a confirmation email. If you do not receive such an email,

your submission was not received. If you have problems with the submission process, send an email to the instructor **before** the due date.