**Project 2: MergeSort**

(Due September 19 at 5 pm)

Learning Objectives:

* Write C++ programs in a modern software development using the CLion IDE and version control with Github repositories,
* Write modular code in a single file organized into functions,
* Apply recursion to sort a vector of integers,
* Implement and manipulate the contents of vectors.

**MergeSort**

Write a program named “mergesort.cpp” that will ask the user for a random seed and a vector length, populate the entries of a vector of that length with random integers, and then sort those entries in ascending order using the mergesort algorithm.

Mergesort is a beautiful (and efficient) sorting algorithm that is most easily implemented using a recursive function. The algorithm works by splitting the list into two equal sized lists, sorting each half individually, and then merging the results back into one sorted list. Here is the pseudocode:

sort(A)

Base case: if A has length 1, quick return

Split A into a left half L and right half R

Recursive call: sort(L)

Recursive call: sort(R)

Merge L and R into A in sorted order

So if the original list is

5 3 2 6 | 1 9 7 8

after the recursive calls each half is sorted:

2 3 5 6 | 1 7 8 9

and after the merge the whole list is sorted:

1 2 3 5 6 7 8 9

The merge operation should be encapsulated into a function that requires some temporary space. It takes as input two sorted lists and outputs the union of those lists in sorted order. Because the input lists are in sorted order, it can be done very efficiently: the idea is that the smallest item in the output is either the smallest (first) item in the first list or the smallest (first) item in the second. This item can be moved from the input list to the output list, and the process can be repeated.

For the example above, the merge process works as follows. After sorting the left and high halves, we have:

In: **2** 3 5 6 | **1** 7 8 9

Out:

and since 1 < 2 we move 1 into the output and consider the next item:

In: **2** 3 5 6 |  **7** 8 9

Out: 1

and since 2 < 7 we move 2 into the output and consider the next item:

In:  **3** 5 6 |  **7** 8 9

Out: 1 2

and since 3 < 7 we move 3 into the output and consider the next item:

In:   **5** 6 |  **7** 8 9

Out: 1 2 3

and since 5 < 7 we move 5 into the output and consider the next item:

In:    **6** |  **7** 8 9

Out: 1 2 3 5

and since 6 < 7 we move 6 into the output and consider the next item:

In:    |  **7** 8 9

Out: 1 2 3 5 6

And since the first list is now empty we copy the rest of the second to the output:

In:    |

Out: 1 2 3 5 6 7 8 9

In the assignment, I have provided you with some meaningful starter code. The main function is nearly complete, serving as a unit tester of the MergeSort function. Your tasks are to:

1. Write a MergeSortedLists function using the specified signature (i.e. function prototype),
2. Write at least one unit test (see Section 6.5 in the zyBook) of MergeSortedLists inside main,
3. Write a recursive MergeSort function using the specified signature,
4. Call your MergeSort function to complete the MergeSort unit test.

Tips

* Commit and push frequently, and use descriptive commit messages.
* Complete the tasks in order, making sure to catch all bugs in MergeSortedLists before working on MergeSort.
* Use temporary string indentAmt parameters to help make output statements for debugging recursive functions more interpretable (see Section 7.4 in the zyBook).

Project 2 Invitation Link: <https://classroom.github.com/a/m58oxaZO>

**Implementation Notes:**

Any program that does not compile without errors (warnings okay) will not be graded and be given an automatic 0 points for that part.

**Comments and Style:**Although there will be no formal policy on commenting and style, the reader should be able to easily follow the main purpose of the code. Each block of code, including each function, that does something significant must be commented. The variable names should be easily recognizable and acronyms should be avoided if possible.

**Project Submission:**

You will develop your program for this project with CLion connected to a remote Github repository given to you. Points will be deducted for an incorrect filenames and no credit will be given for late submissions.

**Pledged Work Policy:**

Assignments in Computer Science courses may be specified as "pledged work" assignments by the professor of the course. When an assignment is specified as "pledged work" the only aid that the student may seek is from either the course professor or TAs (including CS Center tutors) that the professor has explicitly specified. On "pledged work" assignments the student may not use the services of a tutor.

You may discuss only basic C++ syntax and general computer science concepts with everyone else. Any other communications of the project (e.g., giving your code to someone else or seeing someone else’s code) are strictly prohibited except with the professor and TAs of the course. Your code and your implementation of the project must be the product of your own work.