

CST8130: Data Structures

Lab 3: Using Recursion and MergeSort

Introduction

I have supplied an executable and source code that outlines an implementation of *MergeSort*¹. It is intended to sort *int* values. The recursive *MergeSort()* function is currently empty; you need to craft the few lines of code that implements the recursive algorithm.

Following the successful implementation of the *MergeSort* algorithm, you will implement *MergeSort* to sort one index in your *EarthQuakeDataSet*.

Detailed Steps: Build Program

Begin by downloading the code and building a project.

Look to the online lecture on *MergeSort* to review the recursive part of the algorithm (only about 5 or 6 lines inside the recursive function)². The program will read all records (over 157,000), but do nothing with them. After compiling the program, run it under the debugger, set a breakpoint at the start of the *MergeSort()* function you implement. You'll want two different inspector windows:

- *Call Stack*: This window will give a clear picture of the pattern of recursive calls.
- *Watch Window*: Inspect the two arrays of *int* values: *panArrayOfNumbers* and *panWorkSpace*. Remember, because of the declaration of these variables, they will only show the first element, by default. You can view more items by adding a comma and size in the *Watch Window*.

Detailed Steps: Implement MergeSort with Earthquake Data

Port the *MergeSort* algorithm from the first part of this lab to the program that sorts your *EarthQuakeRecords*.

So, what needs to change?

You will be sorting an array of pointers to

EarthQuakeRecord objects³, instead of an array of *int* values.

When comparing items, you'll access a data member in an *EarthQuakeRecord* object.

Detailed Steps: Memory Map

Your memory map will be very similar, but you'll need to add the array of temporary values that are used during the *MergeSort* process.

Submission Requirements

You will need to include the following:

- Problem Statement.
- Test Plan that validates the correctness of your work. Here, your test plan should clarify how you intend to use the debugger to inspect memory-resident data and the call stack. Later you'll use the display function to view data. You may also need to set breakpoints at strategic locations.
- Source Code (with extensive comments that convince me that you understand your work).
- Memory Maps to clarify how memory will be organized based on your implementation.

Note: No PDL is required for this lab. The key to understanding this work is your memory map, and your casting and dereferencing operations.

¹ The executable is complete; the source code is intentionally incomplete.

² I've already completed the *Merge()* function for you. You need only implement the recursive *MergeSort()* function.

³ Of course, an array of pointers is really an array of integer values, since a pointer is just an integer that we choose to interpret as an address.