Exercise – Merge Sort

Introduction:

In this exercise we will write the *merge sort* algorithm and profile its performance.

This tutorial follows on from the *insertion sort* tutorial. If you have not yet completed that tutorial you may want to do that before completing this one. While it is not essential to have completed that tutorial, only the code that needs to be modified will be presented here.

In the exercises for the remaining sorting sessions we will be adding the various sorting algorithms to this application and comparing their performance.

Application Setup:

By this point most of our application should already be set up.

For this exercise you can either keep the existing bubble and insertion sort algorithms, and compare the performance of all three, or simply replace the insertion sort algorithm with merge sort. The former will require slightly more code, but will give you a good idea of how all of the sorting methods compare.

Add the following code (that will call and monitor the execution time of the merge sort function) to your main function:

// Profile Merge Sort

memcpy(valuesToSort, values, sizeof(int)\*size);

t1 = high\_resolution\_clock::now();

mergeSort(valuesToSort, 0, size-1);

t2 = high\_resolution\_clock::now();

for (int i = 0; i < size; i++) {

std::cout << valuesToSort[i] << ", ";

}

std::cout << std::endl << std::endl;

std::cout << "Merge Sort took " << (t2 - t1).count() <<

" nanoseconds" << std::endl;

The only thing you will need to be aware of is that the input arguments for *mergeSort()* are slightly different from both bubble and insertion sort.

The *mergeSort()* we need to pass in the start and end index of the array (0, and size-1), as opposed to just passing the size of the array.

The function declaration for merge sort is as follows:

void mergeSort(int\* const array, int startIdx, int endIdx);

You will also need the *merge()* function (which combines two sub arrays):

void merge(int\* const array, int startIdx, int midIdx, int endIdx)

Exercise:

Complete the merge sort algorithm.

You may want to refer to the pseudo-code and notes included in the lecture slides for this session.

Be aware that the *p*, *q*, and *r* variables in the pseudo-code listed in the lecture slides refer to the *startIdx*, *midIdx* and *endIdx* variables respectively in the code segments above.

After you complete the merge sort algorithm and profile it’s performance, what do you notice about its execution time in relation to bubble and insertion sort?

Modify the size of the array containing the unsorted data so that merge sort becomes the more efficient algorithm.