Exercise – Quick Sort

Introduction:

In this exercise we will write the *quick 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 quick sort function) to your main function:

// Profile Quick Sort

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

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

quickSort(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 << "Quick Sort took " << (t2 - t1).count() <<

" nanoseconds" << std::endl;

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

The *quickSort()* 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 quick sort is as follows:

void quickSort(int\* const array, int p, int r);

You will also need the *partition()* function:

int partition(int\* const array, int p, int r);

Exercise:

Complete the quick sort algorithm.

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

Be careful in your conversion of pseudocode to c++. For example, the loop in the partition function is inclusive, so be sure the end condition in the for loop is “*j <= r – 1*”.