Csci 335 Assignment 4

Due Friday, Nov. 22

**Programming: Using and Comparing Sorting implementations (100 points)

In this assignment you are going to compare various sorting algorithms. You will also modify the algorithms in order for a Comparator class to be used for comparisons.

To **start** you should write a small function that verifies that a collection is in sorted order:

```
template <typename Comparable, typename Comparator>

bool VerifyOrder(const vector<Comparable> &input, Comparator
less_than)
```

The above function should return true iff the input is in sorted order according to the Comparator. For example, in order to check whether a vector of integers (vector<int> input_vector) is sorted from smaller to larger, you need to call

```
VerifyOrder(input_vector, less<int>{});
```

If you want to check whether the vector is sorted from larger to smaller you need to call

```
VerifyOrder(input_vector, greater<int>{});
```

The **next step** is to modify the code of heapsort, quicksort and mergesort, such that a Comparator us used. For example, the signature for quicksort will become:

```
template <typename Comparable, typename Comparator>

void QuickSort(vector<Comparable> &a, Comparator less_than);
```

Modify the code provided with the book for these algorithms. They are included in file Sort.h under directory CodeFromBook

Part 1 (60 points)

In order to check the running time of the three algorithms, create a driver program that will be executed as follows:

./test_sorting_algorithms <input_type> <input_size> <comparison_type>,

where <input_type> can be random or sorted, <input_size> is the number of elements of the input, and <comparison_type> is either less or greater.

For example, you can run

```
./test_sorting_algorithms random 20000 less
```

The above will produce a random vector of 20000 integers, and apply all three algorithms using the less<int>{} Comparator.

You can also run

```
./test_sorting_algorithms sorted 10000 greater
```

The above will produce the vector of integers containing 1 through 10000 in that order, and will test the three algorithms using the greater<int>{} Comparator.

Generation of a random vector is included in the sample code provided.

Suppose that you execute

./test_sorting_algorithms random 20000 less

The output should look as follows:

```
--------

Running sorting algorithms: random 200000 numbers less

HeapSort: Runtime: X ns

          Verified: 1 (<- this should be the output of the
VerifyOrder() function to be executed after the end of the
sorting procedure).

MergeSort: <Same as above>

QuickSort: <Same as above>
```

For how to time the routines see the sample code provided (function TestTiming()).

Part 2 (40 points)

Test some variations of the quicksort algorithm. Investigate the following pivot selection procedures:

a) Median of three (as in the slides)
b) Middle pivot (always select the middle item in the array)
c) First pivot (always select the first item in the array)

The executable should run using the same parameters as in Part 1:

./test_qsort_algorithm <input_type> <input_size> <comparison_type>

Note that the code for Median of Three (provided with the book) needs to be slightly modified for Middle and First.

If for example you run it as

```
./test_qsort_algorithm random 20000 less
```

Then output should be of the form:

```
Testing quicksort:  random 200000 numbers less

Median of three

Runtime: X ns

Verified: 1

----

Middle

Runtime: X ns

Verified: 1

----

First

Runtime: X ns

Verified: 1

----
```