CS 435, Online Algorithms	**Programming Assignment 2 Sorting Algorithms**	Dr. David Nassimi

This programming assignment is to implement the following three sorting algorithms and to collect time performance measurements on them.

1. Mergesort,
2. Heapsort,
3. Quicksort.

The running time of each sorting algorithm must be measured in two ways:

- Count the number of key comparisons, COMPCOUNT. To obtain this count, it is suggeted that you write a function COMPARE(X, Y), which will perform a comparison between X and Y and increment COMPCOUNT. Then, wherever you need to perform a key comparison in your algorithms, you simply make a call to this function.
- Use the actual measured CLOCK time.

You need to carry out the experiment in two parts.

1 A small array size, $n = 32$, to verify correctness

Run each algorithm on three sets of data:

(1) Sorted; (2) Reversely sorted; and (3) Randomly generated.

For simplicity, you may use integers. For each case, make sure the same original data is input to all three algorithms. Print both the sorted array and measured number of comparisons for each case to show the correctness of your algorithms.

2 Large array sizes to determine time complexity

Run each sorting algorithm on the following array sizes:

- $n = 2^{10} = 1,024$
- $n = 2^{15} = 32,768$
- $n = 2^{20} = 1,048,576$

For each value of n, produce an array of n randomly generated elements only once. (For simplicity, you may use integers.) Again, make sure you provide the same randomly-generated array of data to all three sorting algorithms.

Tabulate the performace results, listing both the number of comparisons and the measured clock times. Use the tabulated number of comparisons to determine the time complexity. For example, does the data for quicksort verify $O(n^2)$ or $O(n \log n)$ time, and what is the constant factor? Note that since you use a random sequence to sort, the running times should correspond to the *average-case* time complexities.