The Assignment:

You will implement, test and measure the performance of the following sort functions:

REQUIRED Sorts

Bubble Sort	3 Points
Optimized Bubble Sort	1 Point <= Easiest Point EVER
Insertion Sort	3 Points
Selection Sort	3 Points

PICK ANY TWO Fast Sorts

Quick Sort with or without Recursion	5 Points
Merge Sort w/ or /wo Recursion	5 Points
Heap Sort w/ or /wo Recursion	5 Points
Radix Sort (PG 681-2, Q10-Q11)	6 Points

EXTRA CREDIT, 5 Points Maximum:

Each Additional Fast Sort	+2 Points Each
Any Sort w/ or /wo Recursion	+1 Points Each
Multi-Thread Any Recursive sort	+3 Points Each <= Quick Sort is not too hard!

Purposes

Ensure that you can understand the implementations and trade-offs of various sorting algorithms.

Before Starting

Read all of Chapter 13.

File that you must write

sort.cpp

: This file contains a shell and test harness for your sort functions. Each sort should have it's own function. It will sort the array in-place (The input array must have your final answer when the function returns). Helper functions are Not allowed. Several sorts, including a Shaker Sort were provided as examples of the interface to your functions and various techniques used to implement and optimize these sorts.

The test harness will run all algorithms, compare their execution time for unsorted, mostly sorted (99.9% sorted) and fully sorted lists of numbers (int). Note that integers can be both Positive AND Negative! The assignment was constructed so that Each algorithm gets the exact same list of numbers (Same values and order).

In addition there is a helper function to print the list:

```
void printArray (int*array, size_t count) ;
```

Sort functions will be of the following form:

```
void NameOfSort(int*array, size_t count) {
                        // HINT:  size_t is an unsigned int!
     if(count <= 1)
     return; // No work for an empty or 1 element array.


   // Student Code Goes Here.
```

```
    return;
}
```

Sort functions Can be added to the test harness, by adding an entry to the yourSortRoutines[]array. This array is nullptr terminated, so make sure that entry remains last.

```
sortRoutine yourSortRoutines[] = {
    {shakerSort, "Shaker Sort"},

   // Student Code Goes Here.

    {nullptr, nullptr}
};
```

The Heap Sort, Merge Sort, Quick Sort and Radix Sort Functions

There will be several functions in your sort.cpp, one for each sort function and associated helper functions.

However, for recursive algorithms, the interface function is not suitable. So the interface function will likely just do some initialization, then call the recursive function to start off your sort process.

Note that for the Radix Sort, you need to deal with Positive and Negative numbers in the Array!

Extra Credit Options

In addition to the standard algorithms, there are some extra credit options.

Complete additional Fast sort algorithms.

Add non-recursive / recursive (opposite of what ever you did in the first part) Sorts. (Hard)

Threaded Recursive Sort. (Medium, but super cool t know this powerful technique)

Testing Considerations
Since there is an existing test harness, your primary test factor is the size of the array to sort. Star Small to see if things work at all! Then make it large enough to show meaningful differences in executions times. In addition, if there are boundary conditions (like an odd vs even for the shaker sort or, a none-power-of-2 size for a merge sort), please include those runs too. Add them all to an output.txt file.

Sample Output SortOutput.rtf

First (of 3) Run(s)

```
CIST 004B Sorting Assignment.  Worth 20 points


Enter the size of Array you wish to sort (not more than
100,000,000): 50000

Do you wish a single run (Y/N): y


Now executing a sample Shaker Sort of 50000 items.

A sample Shaker Sort took 5.107361 Seconds.

Now Executing a sample Shaker Sort of 50000 mostly (99.9%)
sorted items.

A sample Shaker Sort took 0.006847 Seconds.

Now Executing a sample Shaker Sort of 50000 pre-sorted items.

A sample Shaker Sort took 0.000370 Seconds.


Now executing the Bubble Sort of 50000 items.
```

```
. . .

Now executing the Selection Sort of 50000 items.

. . .

Now executing the Insertion Sort of 50000 items.

. . .

Now executing the Merge Sort of 50000 items.

. . .

Now executing the Quick Sort of 50000 items.

. . .

Program ended with exit code: 0
```

Second Run

```
CIST 004B Sorting Assignment.  Worth 20 points


Enter the size of Array you wish to sort (less than 1,000,000):
32768

Now executing the Shaker Sort of 32768 items.

The Shaker Sort took 2.142633 Seconds.

. . .
```

Final Run

```
CIST 004B Sorting Assignment.  Worth 20 points


Enter the size of Array you wish to sort (not more than
100,000,000): 2500

Do you wish a single run (Y/N): n
```

```
Testing every Sort Routine with array sizes from 0 to 2500
numbers.
Starting at:              Mon Apr 29 22:37:32 2019
     0 . . . . . . . . . .Mon Apr 29 22:37:32 2019
  100 . . . . . . . . . .Mon Apr 29 22:37:33 2019
  200 . . . . . . . . . .Mon Apr 29 22:37:33 2019
  300 . . . . . . . . . .Mon Apr 29 22:37:33 2019
  400 . . . . . . . . . .Mon Apr 29 22:37:33 2019
  500 . . . . . . . . . .Mon Apr 29 22:37:33 2019
  600 . . . . . . . . . .Mon Apr 29 22:37:34 2019
  700 . . . . . . . . . .Mon Apr 29 22:37:34 2019
  800 . . . . . . . . . .Mon Apr 29 22:37:34 2019
. . .
2200 . . . . . . . . . .Mon Apr 29 22:43:54 2019
2300 . . . . . . . . . .Mon Apr 29 22:43:57 2019
2400 . . . . . . . . . .
Finished at:              Mon Apr 29 22:43:59 2019

Program ended with exit code: 0
```

Hint for Visual Studio Users

Microsoft Visual Studio may not support the old-style function to create formatted strings for display of time. If your compiler is unhappy, then please ready the

comments in the provided code and make the prescribed changes. If someone has a platform neutral implementation for this, I will give extra credit (contact me)!