CPSC 1160 SECTION 001 INSTRUCTOR: GLADYS MONAGAN

ASSIGNMENT #4: SORTING AND ANALYSIS

DUE DATE WITH BRIGHTSPACE: OCTOBER 19 AT 11:50 PM

THE LEARNING OUTCOMES

- to implement the fast sorting algorithms *Quick Sort* and *Merge Sort*, and to verify that they really are much faster than the *Cocktail Sort*, which is a quadratic algorithm
- to determine the running time complexity of an algorithm based on empirical data

READINGS

- §7.10
- possibly §12.3
- possibly §17.5.1

- chapter 18 on Developing Efficient Algorithms
  - §18.1 - §18.4
  - not covered: Towers of Hanoi analysis §18.4.3
  - not covered: §18.5, (§18.6, §18.7), §18.8, §18.9, §18.10
  - yes Chapter Summary
- chapter 19 on Sorting §19.1 - §19.5
  - not covered §19.6, §19.7, §19.8

IMPLEMENT

1) Implement the *Cocktail Sort* algorithm in a function with the function prototype

```
void cocktailSort(int A[], int n);
```

you have done so for Assignment #4.

2) Implement the *Quick Sort* algorithm in a function with the function prototype

```
void quickSort(int A[], int n);
```

yes, you can use the class notes.

3) Implement the *Merge Sort* algorithm in a function with the function prototype

```
    void mergeSort(int A[], int W[], int n);
```

and `W` as an array of working storage (`W` has the same length as the array A)[1]
yes, you can use the class notes.

You may want to spend some time optimizing the base case of the recursion for `quickSort` and `mergeSort` because we want to make them as fast as we can. It's up to you.

When you are given a function prototype (as an interface for a function), you must use the function prototype given. However, you are free to overload the function to suit your algorithm.

4) Now, using the using the rand() function, generate arrays of random 15 bit or 30 bit random integers for n = 1000, 2000, 4000, 8000, 16000, 32000. Time how long `cocktailSort`, `quicksort` and `mergeSort` take.

To get an accurate measure for the time for small values of `n`, you will need to repeat the experiment `m` times, that is, time how long it takes to sort an array of `n` values `m` times and then divide the total time `T` by `m`.
For example, for `n=1000`, you may need to repeat the experiment `m=100` times so that the total time is at least 1 time unit.
For the fast algorithms, you may need to use `m=1000` or larger, to get an accurate time `T`.

[1] Do not use dynamic arrays the way it's done in the textbook.

5) Now let C(n) be the time it takes to sort n elements for `cocktailSort`.
Let Q(n) be the time it takes for `quickSort` to sort n elements.
Let M(n) be the time it takes for `mergeSort` to sort n elements.

Let your program developed from 1) 2) 3) 4) and 5) produce the table with data as shown below.

n	$C(n)$	$\frac{C(n)}{n^2}$	$Q(n)$	$\frac{Q(n)}{(n \log_2 n)}$	$M(n)$	$\frac{M(n)}{(n \log_2 n)}$
1000						
2000		y				
4000						
8000						
16000						
32000						

So, for instance, the value y in the table above corresponds to the average time it takes to cocktail sort 2,000 integers divided by 2,000^2. In other words, y was computed by sorting with cocktail sort 2,000 integers, running the sorting `m` times and getting the average, and then dividing that average time by 4,000,000

6) Theoretically (as you know from Assignment #4), `cocktailSort` should be O(n^2) whereas `quickSort` for the average case and `mergeSort` should be O(n log$_2$n).

Try to graph for 10 bonus points

a. n vs. C(n)
b. n vs. Q(n)
c. n vs. M(n).

Does your data of the table support the theory that Cocktail Sort is O(n^2) that Quick Sort is O(n log$_2$n) for the average case and that Merge Sort is O(n log$_2$n)?
Justify your answer and come up with some conclusion using your own data and discuss what you can infer from the table, particularly why the third, fifth, and seventh columns give such values.

Note that we are doubling the n in the table above.

And your conclusion should **not** be that quicksort is faster than cocktail sort or than merge sort.
Tell us about what you can see from the third, fifth and seventh columns and why.

TO SUBMIT WITH BRIGHTSPACE AS A SINGLE ZIP FILE:

A) The source code of the programs. Leave in the source code the timing commands so that we can run your code and reproduce your timings. Comment your source code appropriately and modularize as needed. Remember to name your constants.

   If you use code (say from the lectures) for the sorting algorithms, say so i.e. cite where you got the code from.
   The only code that you may include in this assignment that you have not "written from scratch" i.e. that you did not come up on your own, is the code for the sorting algorithms. But you must cite the source properly.

   Do not use dynamic arrays, we have not covered them in the course yet. Use the arrays the way we have learned and remember to use a named constant when declaring them.

   Declare the arrays in the main program, not in the functions.

B) A document with the table of timing data (i.e. the output of your program) **PLUS** a discussion as per 6).
   Include the graphs if you made them (and use them to support your conclusions).
   Do not make up the timings.

Marking Scheme (may be modified slightly)

10	Implementation of Cocktail Sort
12	Implementation of Quick Sort
12	Implementation of Merge Sort
20	Application Program Random numbers Tests as required Timings Display results
16	Results are accurate (table)
10	Analysis and conclusions
20	Programming style (named constants, documentation of the functions, interface matches the given function prototypes, modularization, …)