Timing Tests, Lab Assignment 8

Re-submit Assignment

Due Oct 25 by 11:59pm **Points** 100 **Submitting** a file upload **File Types** cpp and h

PART 1

Perform A Simple Timing Study

Purpose. In this lab you will apply the "4-cycle timing code" presented in the lecture topic 8 notes. The purpose is to prepare you for making big oh determinations and confirming them with timing studies.

Requirements. Write **ParsingWithoutDupChecking.cpp**, applying this module's timing test code to the reading of the **dvc-schedule.txt** (http://cs.dvc.edu/sp2016/comsc210/attachments/dvc-schedule.txt) file ***from lab 5***. Do not read the whole file -- just read n lines. Parse as you did in previous labs that used this same input file, but do not check for duplicates and do not count. Read lines and parse *only*, timing how long it takes to do so for n lines. Start with n = 8000 for the first cycle, and confirm O(n) -- that is, the time it takes to read the file is directly proportional to the number of lines read from the file.

In each of the 4 timing cycles, do this: start the timer, then open the file. Run the EOF loop with duplicate checking and counting *removed*. After the EOF loop ends, close the file and stop the timer. You should not need any of your H files to support duplicate checking and counting, so do not use one.

Note that your computer may cache your input file, throwing off the timing for the first cycle. So run your program more than once to see it work right.

Submit the CPP to the class website for credit. Do NOT submit the TXT input file.

Example.

```
1.436 (expected O(n)) for n=8000
2.742 (expected 2.872) for n=16000
5.442 (expected 5.744) for n=32000
10.828 (expected 11.488) for n=64000
```

Windows 7 Users NOTE: Old versions of Windows (like 7 and prior) do not have fine enough timing resolution to do this exercise properly. The resolution gives results like 0.0, 0.06, 0.12, 0.18, and so on to the nearest 1/16th of a second. So you may want to apply the following technique, like the one used to test fast algorithms (like O(1) and O(log n)) -- put the timed code in a extra inner loop, by adding the **bolded** lines of code:

```
clock_t startTime = clock( );
for (int rep = 0; rep < 10; rep++)
{
  // open the file, read and parse n lines, close the file
}
clock_t endTime = clock( );
```

If that's still too fast, use a larger number than 10 for the inner "rep" loop cycles. ***But take that out before submitting your file!*** Either that or set the #of cycles to 1.

PART 2

Determine Big Oh And Perform Another Simple Timing Study

Purpose. For you to determine big oh for a process and confirm it using the tools provided in this class.

Requirements. Redo lab 4's parsing + duplicate checking, but still no counting of how many courses were offered by subject code! Call it **ParsingPlusDupChecking.cpp**. You'll need one of your H files from past labs for the duplicate checking, so submit the needed H file(s) along with the CPP: **DynamicArray.h** and/or **StaticArray.h**.

HINT. Start with n = 800 for the first cycle -- no need to take minutes for this to run. It will still give good results.

PART 3

Perform A Timing Study On Nested For-Loop Sorting

Purpose. In this lab you will make your own determination for the big oh of nested for-loop sorting of an array, and confirm your determination.

Requirements. Write **NestedForLoop.cpp**, applying the timing test code from the module, to the sorting of a dynamic array of doubles. You decide what you expect the big oh to be, and what n to use for the first cycle. In each cycle, create the array and fill it with random values (using srand and rand). Then start the timer, perform the sort, and stop the timer. *Write code to verify that each number in the array is greater or equal to the number before it, starting with the 2nd number. Use assert, like this:*

```
for (int i = 1; i < n; i++) assert (a[i - 1] <= a[i]);
```

This way, there's not a lot of output to look through in order to verify that the sorting performed correctly.

Use your own **DynamicArray.h** or a regular C++ dynamic array, as you wish. If you use your own, submit its H file along with the timing test CPP (if you did not already do so for part 2).

Lab Assignment Rubric													
Criteria	**Ratings**												**Pts**
Fully accurate results, following all specifications **view longer description**	Works the first time. 70.0 pts	Works on the 2nd try 65.0 pts	Works on the 3rd try 60.0 pts	Works after 4 or more tries. 50.0 pts	Doesn't work after 2 weeks. Partial credit. 20.0 pts	Not submitted within two weeks of the due date. 0.0 pts	Work is not original -- appears to be a marked-up copy of the work of another or previous student. 0.0 pts						70.0 pts
Submits all work on time, fully complete if not fully correct. **view longer description**	Submitted on time 20.0 pts	Submitted on time, but one or more files are missing or not correctly named. 16.0 pts			Submitted on time, but with missing identification in one or more submitted CPP or H files. 15.0 pts			Submitted on time but not fully complete. 10.0 pts		Late or wholly incomplete! 0.0 pts			20.0 pts
Well-organized and professional quality code. **view longer description**	Fully meets expectations 10.0 pts	Mostly meets expectations, just needs to be a bit more careful. 8.0 pts			Many areas are well done, but there are a lot of areas that need work. 6.0 pts			Getting there, but needs to be a lot better. 3.0 pts		Needs a lot of work. See the instructor for guidance. 0.0 pts			10.0 pts
Total Points: 100.0													