Ready? Set. Sort!

First Draft: February 21, 2015 (Saturday) @ 11pm pushed and tagged to Github (v4.0)
Final Submission: March 2, 2015 (A Monday) by 11pm pushed and tagged to GitHub (v4.1)

Introduction

In this project, you'll implement a sort of words from a text file based on the following priorities:

- High Priority: sort by length of words
- Lower Priority: sort alphabetically

Your goal is to develop a sorting algorithm that can sort the words based on the stated criteria as fast as possible. **The competition is based on actual execution time**, not Big-O or Big-Theta analysis. You can incorporate any sorting algorithms that you'd like including ones you discover via research online.

Some important points:

- The input will contain words as sequence of characters. Punctuation may be present, but you do not need to parse it out. Just assume that it is part of the word (example: mark!! would be length 6).
- Implement a class named SortingCompetition. A header file for this class can be found at the end of this document. You must not modify the public interface of this header file. However, you can add any private data or methods you'd like. Certain stipulations are indicated in the header below.
- You must read all of the words from the input file before beginning any part of the sorting. **They must be read into a linear/sequential container such that if output from beginning to end, the original file would be recreated.**
- The TAs will use a standard main driver that will instantiate a SortingCompetition object, send it the names of the input and output files, and then time the sorting method (likely multiple times).
- TAs will gather preliminary run times for each competitor based on the preliminary submissions on Feb 22 (hopefully). Those times will be posted online so you can see where your algorithm run time rates compare to your competitors.
- You may work independently OR in teams of no more than two students (NO EXCEPTIONS, so PLEASE DON'T ASK!)

Grading

Everyone must participate. This is not an optional exercise/competition. You may choose to not refine your algorithms to increase speed, but minimally, you must develop a working solution. The coding portion of the assignment is worth 50 points.

With your final submission, you should submit a 2 - 3 page document containing the following (worth 50 points):

- any and all sorting algorithms that you tried and how you attempted to improve the overall speed of your program with each iteration of your design. You should have tried no fewer than 5 different methods to increase the speed of your program.
- strategy for evaluating your implementations. How did you gather timing data? What data sets did you use (characterize them)?
- give a high-level overview of how you arrived at your final solution. What is interesting about this solution? How does the execution time of this solution change with respect to the size of the input?

Winners' Prizes

The First Place winner(s) can choose one of the following options as prizes:

- Raise a programming project grade to a perfect score as long as a reasonable attempt was made (this is not the same as allowing you to skip a programming project). This does NOT apply to the final project.
- Drop a homework grade.
- 3 points added to your SEMESTER AVERAGE (yes, that's right.... your semester average)!!
- Free dinner with Fontenot and some of the TAs (Fontenot gets veto power over restaurant).

The Second Place winner(s) can choose one of the following options as prizes:

- Drop a homework grade.
- Add 20 points to a programming project final grade (not including the final project).
- Free Pokey-O's or FroYo (or some other kind of dessert) with Fontenot and some of the TAs

The Third Place winner(s) will get the following:

- 10 points added to either a homework grade OR programming assignment of their choice.

Things that will get you Disqualified

These things will get you disqualified from the competition and likely 0/50 points for the programming portion of the project:

- sorting as you're reading from the file
- not conforming to the specification listed below
- attempting to hack the timing mechanism in any way
- other sketchy things (we'll be looking at your source code, ya know!).

Measuring Time

C++ 11, the latest version of C++, contains a timing library called **chrono**. The code below is a complete example of using the chrono lib from http://en.cppreference.com. It determines the amount of time needed to execute the Fibonacci function. The code may initially look confusing, but it is due to the extensive use of namespaces. Take a few minutes and read through it line by line.

```
#include <iostream>
#include <chrono>
#include <ctime>

long fibonacci(int n)
{
	if (n < 3) return 1;
	return fibonacci(n-1) + fibonacci(n-2);
}

int main()
{
	//declare 2 time points
	std::chrono::time_point<std::chrono::system_clock> start, end;

	//store the current time (now()) in start, execute
	//Fibonacci function, then store current time in end
	start = std::chrono::system_clock::now();
	std::cout << "f(42) = " << fibonacci(42) << '\n';
	end = std::chrono::system_clock::now();

	//subtract end from beginning to get number of seconds elapsed
	std::chrono::duration<double> elapsed_seconds = end-start;
	std::time_t end_time =
		std::chrono::system_clock::to_time_t(end);

	//output the duration.
	std::cout << "finished computation at " <<
		std::ctime(&end_time)
		<< "elapsed time: " << elapsed_seconds.count() << "s\n";
}
```

The Sorting Competition Class

Your solution must implement this class. Otherwise, the TAs will not be able to grade your solution.

```
class SortingCompetition
{
private:
//you are free to determine your own internal
//implementation with the following stipulations:
```

```
//1) when the prepare method is called, you must make a
//copy of the original dataset into another instance-level
//data structure which will be used by the sort method. This will
//allow for multiple executions of the sort method in order to
//get better timing data.
//2) your data structure must be linear (no trees).

public:

      //basic constructor that accepts an input
      //file name
      SortingCompetition(const string& inputFileName);


      //change the input file name
      void setFileName(const string& inputFileName);

      //read the data from the file and store it in
      //a linear data structure.
      //No sorting actions can be done in this method.
      //This includes no duplicate removal or anything else
      //that could make your sorting more efficient later.
      //Literally, the 5th word in the file should be
      //in the 5th place in your structure.
      bool readData();

      //copy the data from the original data structure
      //into a second location that will be used for sorting.
      //This will allow you to sort the same data set (with
      //the same starting order of elements) multiple times.
      //You can then calculate the average of execution times for
      //one data set against one algorithm.
      //No sorting actions can be done in this method.
      bool prepareData();

      //sort the data based on the criteria set forth in the
      //hand out.
      //THIS IS THE FUNCTION THAT WILL BE TIMED.
      void sortData();

      //using outputFileName, write the "sorted" data structure
      //to the file. This will be used to test the validity of
      //your sorting algorithm (in other words, did it sort properly?).
      void outputData(const string& outputFileName);

};
```