The Sorting Competition

Due: Monday, March 11, 2019 @ 6am (may be submitted earlier due to Spring Break) pushed to GitHub in the CSE 2341 Repo created by course staff and Release Issued

Note: With Spring Break starting as this project is due, you may be unable to reach a TA the weekend before spring break if you need help. It is ***highly advised*** that you finish the project early!

Introduction

In this sprint you will develop a program that can sort a list of words from a text file. The words must be sorted according to the following conditions:

1. Primary Sort Condition - Sort by length of word (number of characters)
2. Secondary Sort Condition - Sort alphabetically. Since this is a secondary sort condition, it will be applied to words that are all of the same length. So, all 2 letter words would be sorted alphabetically, all 3 letter words would be sorted alphabetically, and so on.

Implementation Details

- The project is to be completed individually.
- The input to your program will be one file.
  - The first line of the file will contain an integer representing a count of the number of words in the file.
  - The second line of the file will be a count of the number of words which you need to output from the sorted list of words.
  - Each word will be made up of only printable characters (no spaces, no punctuation, etc.) and will not exceed 30 characters.
    - For example, if the first line were 10 and the second line were 2, you would need to find the smallest 2 items from the 10 in the list. (It's possible that sorting the entire list just to find the top few is very inefficient!)
- The output of your program will be one file. The file will consist of a sorted list of words from the input file, each word on a separate line
- You'll perform case-sensitive sorting. cat is different from CaT.
- While you don't have to use **strcmp**, for questions of "alphabetic ordering", the answer would be based on however strcmp would order two strings.
- The dataset may contain duplicate words. Duplicates shall not be removed.
- If you use a container class, you must use your vector class. Any other container classes must be approved by Dr. Fontenot or Head TA Jake Carlson. You are permitted to use primitive arrays/pointers, however.
- Any sorting algorithm used in this project must be implemented from scratch by you.

    - You are encouraged to read about and research various sorting algorithms. But you need to fully implement them yourself. In other words, you need to research them to the point you understand them enough to write code for them.
    - You may NOT use any sorting methods in the standard library (including but not limited to std::sort or an ordered container)
- All solutions must be single threaded.
- No CATCH tests are required for this project.

The spirit behind these stipulations is to prevent the use of any standard library functionality that already implements sorting. We want everyone to go through the process of researching, testing, and iterating on their own implementations.

Writing the Sorter

Your sorting program must accept two command line arguments, one with the name of the input file of words, and the other with the name of the output file of sorted words:

```
./sorter <InputFile> <OutputFile>
```

How you design your sorting program is up to you! Your only goal is to make it work as fast as possible while maintaining good standards in coding.

We will be timing your sorting algorithms by measuring the CPU time it takes to execute your entire program (from reading in the data all the way to writing it out to a file). Every optimization you can make along the way will work in your favor. We will use an OS-level tool to measure the time it takes for the following command to run:

```
./sorter <InputFile> <OutputFile>
```

You can test your own timing using the "time" command in the VM:

```
time ./sorter <InputFile> <OutputFile>
```

The output of this command will be three numbers. The one you care about is the "**user**" time.

We will use a number of test datasets, varying in length from 10 elements to 10 million elements. You should also test your program against data sets of various lengths.

Disqualifiers

These things will get you disqualified from the competition:

- not conforming to the specification listed herein
- attempting to hack the timing mechanism in some/any way
- other sketchy things that violate the spirit of the exercise (we'll be looking at your source code, ya know!)

Prizes

First Place: Add 50 points to a PA 01 or PA 02 OR dinner with Fontenot and some of the TAs at the end of the semester.
Second Place: Add 20 points to a PA 01 or PA 02
Third Place: 10 points added to a PA 01 or PA 02