CS214 Project 0: Simple CSV Sorter Documentation
Due Date: 9/30/2018

Revanth Korrapolu
Andrew Russomagno

Design:

The program reads the input file and makes a 2-dimensional Row size x Column size database array to hold each cell of information in it's own cell in the array and simulate the table. We then made a second one-dimensional array of structs that holds a int rowid value and a "sorting value" which is the value of the movie in the sorting column. Those structs have two fields, the processed sorting value (removing leading/trailing whitespace) and their rowID. We then send that array into the mergesort and receive the sorted array back. As we traverse this sorted struct array, we get each's original rowID and output all of that row's values from the database array. Our mergesort sort also accounts for comparisons from both string and numeric values. When deciding how to rank the columns, we used the default ASCII numbers to compare strings and floats to compare numeric values. Also, we decided to place null values at the top during an ascending order.

Assumptions:

An assumption we had originally used was that it would be easier to have all the items be strings for comparison reasons and universal use. That gave us a little hiccup when it came to sorting numeric columns which is explained in more detail below.

Difficulties:

There were a few difficulties that we ran into while making the program. First one happened when it came to parsing the lines. We had used strtok to tokenize it based on the comma's but the problem we hit with that was it didn't evaluate the null spaces that were scattered in the metadata properly. Also, the commas that existed within the titles of movies would through of the tokenizer since it uses the same delimeter. We resolved this by creating our own tokenizer and parsing strings one char at a time. Another wall we hit was we kept getting "Segmentation Fault: 11". We had made small errors that were causing us to have bounding errors. Another difficulty we had was when sorting we made everything a string for easier comparison but then the sorting prioritized capital letters above lowercase so the lowercase was sorted after all the uppercase. None of our attempts to make it all lowercase for comparison purposes would work but we were able to come across of string comparison function, called strcasecmp, that compared regardless of whether it's upper or lowercase. The last noted issue came with the previous one that when we wanted to sort numeric columns because since we had made them strings then it would sort by the first digit then sort that group by the second digit and then so on. We had to make a special check to tell if a column was all numbers to convert to float or only partially composed of numbers, like some of the movie titles. For example, the movie titled “1776” was still saved/compared as a string because other values in that column could only be saved as strings. There were several other abnormalities that were throughout the data that we came across through testing. One was that not every cell was filled and there were scattered null spaces throughout which found because it gave us an error. Another was that a portion of the movie titles had extra spaces at the front of their strings which we saw during our sorting of it since they would get grouped together.

<u>Testing Procedure</u>:

Once we felt like we had a good base of our code we would test constantly throughout with different changes to keep checking if it would work. To avoid wasting time having it do the whole huge file to test little changes we had made a separate test file with about 20 lines that we had taken from the original file. That way it processed much quicker. Once we got through the basics and work on full rows we then made another test file that contained lines that had abnormalities from the original file like null cells or strange characters. That’s how we found that we had a null space problem and were able to isolate it. When we thought it was sufficient enough with small files we got it going and able to work with the large original file. Overall, we tested for edges cases, null values, and incorrect/insufficient arguments. Are program handles these errors gracefully.

<u>How To Run</u>:

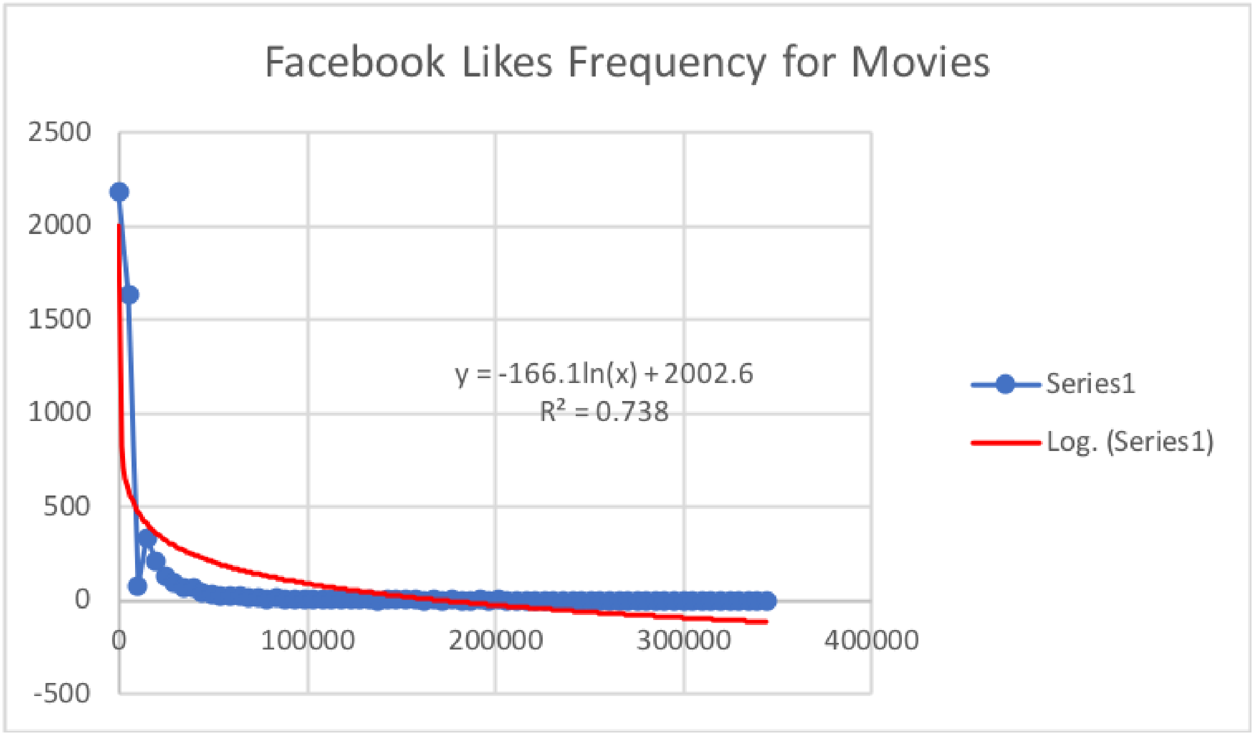
You can run it like it is specified to with the following code, “cat input.file | ./simpleCSVsorter -c movie_title > sortedmovies.csv”. However, in order to achieve the extra credit you can input any CSV but in the following format, “cat input.file | ./filename -c movie_title -r ### > sortedmovies.csv”, where ### is the number of rows in the new file. (ex: cat input.file | ./filename -c movie_title -r 5044 > sortedmovies.csv)

<u>Header File</u>:

Our header file consists of the declaration of our additional methods and functions from our mergesort and simpleCSVsorter files. It also contained the declaration of our Node struct that held our sorting column’s information.

<u>Extra Credit</u>:

1). We collected the frequencies of the various numeric outputs and wanted to compare them to see if it resulted in a discrete power-law probability distribution with Zipf's law. We accomplished this by creating an algorithm split the numeric values into buckets of size N. In this case, the movie database had 5044 rows. We looked at the facebook likes distribution and those values ranged from 0 to 349000. So our $N = 349000/5044 \sim 70$ buckets. This code for this We took the frequencies of each of those buckets and we plotted them in an excel graph (see below). We discovered that the facebook likes of movies did follow Zipf's law.



*the code for this part is commented out at the bottom of the main method

2). We have programmed our script to handle have any CSV entered into it. To do so, we had it make our database array dynamically sized based on the input file (# of rows/columns). We then also automatically keep all the data as strings in order to have a general comparison checking process. We also have a general numeric checking portion that when going down a column if there are only numeric values found then it turns them all from strings to numbers.