Fall 2016
Due 11/09/2016

CSCI 58000
Program 2, Part 3

This program concerns a new sorting algorithm known as Columnsort. Columnsort works as follows. It begins with n = r*s values placed in an $r \times s$ matrix (2-D array – you'll need to look up somewhere how to handle 2-D arrays in C++), where

$$r \bmod s = 0$$

and

$$r \geq 2(s-1)^2$$

The values are stored in column-major order; that is, running from top to bottom in column 1, then top to bottom in column 2, etc. At the end of the algorithm, the $r \times s$ matrix will be sorted if the values are read in column-major order.

Columnsort involves 8 steps. Whenever sorting is called for, use the general sorting algorithm you used for Part 2.

1. Sort the values in each column.

2. "Transpose" the matrix by reading s values at a time in column-major order and putting them back into the matrix in row-major order. (No doubt you will need extra storage to do this – Columnsort is not space efficient.)

3. Sort the new columns.

4. Reverse step 2 – access in row-major order and insert in column-major order.

5. Sort the new columns.

6. Access the values in column-major order and shift the values down (and up to the top of the next column) by $\left\lfloor \frac{r}{2} \right\rfloor$ positions. You will need to create one additional column to accommodate this shift. Fill the first half of the first column (now empty) with the value $-\infty$ and the bottom half of the new column with the value $+\infty$. Use constant int values -32766 and 32767 to represent $-\infty$ and $+\infty$, respectively.

7. Sort the new columns.

8. Reverse the shift of step 6, that is, access the values in column-major order and shift them up by $\left\lfloor \frac{r}{2} \right\rfloor$ positions. The extra column will go away and the result will be values that, in column-major order, are sorted.

Load one of the three 100000 random files you created in Part 1 in column-major order into a matrix (of type *short int*); use the string filename "*Part1Data.txt*". You must use a non-trivial r and s combination, i.e., you can't just work with a 1-D array of size 100,000 where r = 100,000, s = 1. Sort the matrix in increasing order using Columnsort. Use a variation of your *SortCheck* function from Part 2 as a final check that you have indeed sorted the original data. Then write the sorted results to an output file called *Results.txt*, one number per line.

As before, use the Timer class to record the actual wall clock time of Columnsort itself to the nearest millisecond; do not count the time needed to read the values from the data file into the array nor the time to run *SortCheck* or to output the sorted results back to a file. Also output (to the console) the elapsed time and record that information somewhere.

Do the same for the other two files from Part 1.

What you want to **turn in on Canvas** is your code for Part 3, namely

utility.h
Columnsort.cpp

plus any other .cpp or.h files your program may use, although I'm not expecting anything more.

What you want to **turn in to the TA** (on paper) is a summary of the data gained from this experiment, namely a filled-in table of the following form:

Dimensions	r =	s =
General sort used		
	General sort time	Columnsort time
Trial 1		
Trial 2		
Trial 3		

plus a short paragraph on whether Columnsort seems to be more efficient than sorting the entire array by means of your general sorting algorithm, that is, was Columnsort worth the effort?