Fall 2016

CSCI 58000

Program 2, Part 2

Write a program that will read a data file of integer data, one number per line, and sort that file using one of the general sort techniques we have discussed (your choice of insertion sort, selection sort, mergesort, heapsort, quicksort) . Write the sorted results to a file (your choice of filename, but do not write over the original data file.).

**Do this process 3 times (use the same sorting algorithm for all 3 times), once for each of the three random-integer files you produced in Part 1, so in the end you have 3 different sorted files of 100,000 integers each.**

Details:

1. Read your file into an array of type *short int*. Data type *short int* uses 2 bytes to store an integer, so the maximum value for type *short int* is 32767, and your numbers do not exceed that limit.

2. We've already analyzed the work done by any of these sorting algorithms, but this program is in the nature of an experiment to gather empirical data. So you want to keep track of the wall-clock time your program uses. For this, you can make use of the system clock by creating a timer class, as shown here:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TIMER CLASS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class Timer

{

public:

Timer();

//constructor - turns on the timer

double elapsed\_time();

//compute elapsed time between start and stop

void reset();

//restarts the timer

private:

clock\_t start\_time;

//type of value returned by system function clock()

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END TIMER CLASS DEFINITION\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*Implementation Timer class functions\*\*\*\*\*\*\*\*\*\*\*\*/

Timer::Timer()

//constructor - turns on the timer

{

start\_time = clock();

}

double Timer::elapsed\_time()

//compute elapsed time between start and stop

{

clock\_t end\_time = clock();

return ((double) (end\_time - start\_time))/((double)CLK\_TCK);

}

void Timer::reset()

//restarts the timer

{

start\_time = clock();

}

Count just the time to sort, do not count the time needed to read the values from the data file into the array nor the time to output the sorted results back to a file.

3. Before you write to the file, use a final check to be sure your values are indeed sorted, something like

bool SortCheck(array[ ])

//precondition: items have supposedly been sorted

//postcondition: returns true if sorted, false if not

4. Output (to the console) the elapsed time for each of your three runs, and record that information somewhere. Use the following format statements (before you write the output) to get output to 3 decimal places:

cout.setf(ios::fixed);

cout.setf(ios::showpoint);

cout.precision(3);