ECE368 Project #1

Due Thursday, Feb 13, 2014, 11:59pm

Description

This project is to be completed on your own. You will implement Shell sort using insertion sort and selection sort for sorting subarrays. You will use the following sequence for Shell sort:
$\{1, 2, 3, 4, 6, \ldots, 2^p 3^q, \ldots, 3^d\}$, where 3^d is the largest powers of 3 that is smaller than the size of the array to be sorted. Note that the integers in this sequence can always be used to form a triangle, as shown in the lecture notes.

Functions you will have to write:

All mentioned functions and their support functions, if any, must reside in the program module `sorting.c`.

The first two functions `Load_File` and `Save_File`, are not for sorting, but are needed to transfer the integers to be sorted to and from a file.

```
long *Load_File(char *Filename, int *Size)
```

The file contains $N+1$ integers, one in each line. The first line of the file indicates the number of integers to be sorted, i.e., N. `*Size` should be assigned N at the end of the routine. The subsequent lines contains (long) integers.

```
int Save_File(char *Filename, long *Array, int Size)
```

The function saves `Array` to an external file specified by `Filename`. The output file and the input file have the same format. The integer returned should be the number of (long) integers in the Array that have been successfully saved (or printed) into the file.

```
void Shell_Insertion_Sort(long *Array, int Size, double *N_Comp, double *N_Mov
void Shell_Selection_Sort(long *Array, int Size, double *N_Comp, double *N_Mov
```

Each of the two functions take in an Array of long integers and sort them. Size specifies the number of integers to be sorted, and `*N_Comp` and `*N_Move` contain the number of comparisons and the number of moves involving items in Array. The function with the word "Insertion" in the name uses insertion sort to sort each subarray. The function with the word "Selection" in the name uses selection sort to sort each subarray.

A comparison that involves an item in Array, e.g., temp $<$ Array[i] or Array[i] $<$ temp, corresponds to one comparison. A comparison that involves two items in Array, e.g., Array[i] $<$ Array[i-1], also corresponds to one comparison. A move is defined in a similar fashion. Therefore, a swap, which involves temp = Array[i]; Array[i] = Array[j]; Array[j] = temp, corresponds to three moves. Also note that a memcpy or memmove call involving n elements incurs n moves.

```
int Print_Seq(char *Filename, int Size)
```

For an array of size Size, this function prints the sequence in the file Filename. *The format should follow that of the input file, i.e., the number of integers in the sequence in the first line,*

followed by the integers in the sequence, one in each line. The sequence should start with 1. The sequence should be printed in the order in which the integers appear in the triangle from top to bottom and from left to right (i.e., not sorted). However, you should not to print the sequence in the triangular form; there should be only one integer per line. The function returns the number of integers in the sequence. *Note that it is not necessary to call this function before performing various forms of Shell sort.*

You have to write another file called sorting_main.c that would contain the main function to invoke the functions in sorting.c. You should be able to compile sorting_main.c and sorting.c with the following command (with an optimization flag -O3):

```
gcc -Werror -Wall -Wshadow -O3 sorting_main.c -o proj1
```

When the following command is issued,

```
./proj1 i input.txt seq.txt output.txt
```

the program should read in input.txt to store the list of integers to be sorted, run Shell sort with insertion sort, print the sequence to file seq.txt, and print the sorted integers to output.txt. The program should also print to the standard output (i.e., a screen dump), the following information:

```
Number of comparisons:   AAAA
Number of moves:  BBBB
I/O time:   CCCC
Sorting time:   DDDD
```

where AAAA, BBBB, CCCC, and DDDD, all in %1e format, report the statistics you have collected in your program. AAAA and BBBB refer to *N_Comp and *N_Move, respectively. CCCC should report the time it takes to read from input.txt and to print to seq.txt and output.txt. DDDD should report the time it takes to sort.

The function that you may use to keep track of I/O time and Sorting time is clock(), which returns the number of clock ticks (of type clock_t). You can call clock() at two different locations of the program. The difference gives you the number of clock ticks that pass by between the two calls. You would have to divide the difference by CLOCKS_PER_SECOND to get the elapsed time in seconds. There are typically 1 million clock ticks in one second.

The following command will use Shell sorting with selection sort:

```
./proj1 s input.txt seq.txt output.txt
```

Report you will have to write:
You should write a (brief, at most a page) report that contains the following items:

- An analysis of the time- and space-complexity of your algorithm to generate the two sequences (not sorting).

$$Load\ File\ =\ Size\ +\ 1 \qquad Space\ Size\ (Malloc)$$
$$Save\ File\ =\ Size\ +\ 1 \qquad Save\ File\ Size$$
$$Print\ Seq\ =\ \sum_{i=1}^{?} 1 + x$$

- A tabulation of the run-time, number of comparisons, and number of moves obtained from running your code on some sample input files. You should comment on how the run-time, number of comparisons, and number of moves grow as the problem size increases, i.e., the time complexity of your routines.

- A summary of the space complexity of your sorting routines, i.e., the complexity of the additional memory required by your routines.

Each report should not be longer than 1 pages and should be in PDF, postscript, or plain text format (using the textbox in the submission window). The report will account for 10% of the overall grade of this project.

Please note that Shell sort with selection sort will be quite inefficient if you do not optimize the selection sort. You may not be able to run your program on the largest test case. You should include in your report the reason(s) for the inefficiency of Shell sort with selection sort.

Grading:

The project requires the submission (electronically) of the C-code sorting.c and sorting_main.c and any header files you have created through Blackboard. You also have to submit the report through Blackboard.

The grade depends on the correctness of your program, the efficiency of your program, the clarity of your program documentation and report. It is important all the files that have been opened are closed and all the memory that have been allocated are freed before the program exits.

Given:

We provide sample input files for you to evaluate the runtimes, and numbers of comparisons and moves of your sorting algorithms.

Getting started:

Copy over the files from the Blackboard website. Any updates to these instructions will be announced through Blackboard.

Start sorting!

3