Back up your code regularly!

ECE 368 Summer 2018
Programming Assignment #1
Due on June 25, 2018, 11:59 pm

Description:

This project is to be completed in your own. You will implement Shell Sort using Insertion and Selection Sort for sorting subarrays. You will use the following sequence for Shell Sort:

$$\{1,2,3,4,6,\dots,2^p 3^q\} \quad p,q \geq 0, \qquad 2^p 3^q \leq N$$

Where every integer in the sequence is of the form $2^p\ 3^q$, and is smaller than N, the size of the array to be sorted. Note that most of the integers in this sequence, except perhaps for some, can always be used to form a triangle. There may be incomplete rows of integers below the triangle. For example, if there are 15 integers to be sorted, the corresponding sequence $\{1,2,3,4,6,9,8,12\}$ would be organized as follows, with an incomplete row containing the integers 8 and 12 in the sequence:

1

2 3

4 6 9

8 12

You will submit 3 code files along with a report: **sorting.c, sorting.h, main.c, report_username.pdf** (username is your account name, for example report_aguilarh.pdf)

Functions you will have to write for sorting.c:

All mentioned functions and their support functions, if any, must reside in the program module **sorting.c**. The declarations of these functions should reside in the include file **sorting.h**. These functions should not call each other (unless you implement any such function as a recursive function). They should be called by the main function from your file named **sorting_main.c.**

The first two functions **Load_From_File** and **Save_To_File**, are not for sorting, but are needed to transfer the long integers to be sorted from and to a file in text form, respectively.

long *Load_From_File(char *Filename, int *Size)

The file specified by *Filename* contains N+1 long integers, one in each line. The first line of the file indicates the number of long integers to be sorted (N). **Size* should be assigned to N at the end of the routine. The subsequent lines contain the N long integers to be sorted.
If the input file is empty, the array of size 0 should still be created and **Size* be assigned 0.

Back up your code regularly!

int Save_To_File(char *Filename, long *Array, int Size)

The function saves Array to an external file specified by Filename in text format. 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 into the file.
If the size of the array is 0, an empty file should be created.

void Shell_Insertion_Sort(long *Array, int Size, double *N_Comp, double *N_Move)

void Shell_Selection_Sort(long *Array, int Size, double *N_Comp, double *N_Move)

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* should store the number of comparisons and the number of moves involving items in Array throughout the entire process of sorting. 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 to sort each subarray.

You have to decide the order in which you use the integers in the sequence to perform Shell Sort.

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 memmpve call involving n elements incurs n moves.

int Print_Seq(char *Filename, int Size)

For an array whose number of integers to be sorted is *Size*, this function prints the sequence to be used in Shell Sort as defined earlier in the file *Filename*. Each integer in the sequence should be printed in the order from top to bottom, and from left to right in each row. For example, for the sequence given earlier, the output file should look like:

1
2
3
4
6
9
8
12

There should be only one integer per line. The function returns the number of integers in the sequence.

Back up your code regularly!

Note that when Size is 0 or 1, the output file should be empty because the sequence should be empty and return value should be 0.

You are allowed to write other helper functions in sorting.c. For any of these helper functions, the name should not start with an underscore '_' because the functions that the instructor writes to test your code will start with an underscore '_'. These helper functions should be declared and defined in sorting.c. It is best that these helper functions be declared as static functions.

Function main:
I will provide a file called **sorting_main.c** that would contain the main functions to invoke the routines 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 -O3 sorting.c sorting_main.c -o proj1

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

The program should read in input.txt to store the list of integers in to be sorted in an array, run Shell Sort with insertion sort on the array, print the sequence to file seq.txt, and save the sorted integers in the array in output.txt. The second argument can either be 'i' for insertion sort or 's' for selection sort.

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 are all in %le 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 report the time it takes to sort.

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 sequence (not sorting):

Back up your code regularly!

- 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 comparison, and number of moves grow as the problem size increases, i.e., that 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 page and should be in PDF format. The report will account for 10% of the overall grade of this project. If it is not in PDF format you will receive 0 points for this portion.

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 reasons(s) for the inefficiency of Shell sort with selection sort if your implementation is inefficient.

Submission and Grading:

The project requires the submission (electronically) of a zip file called proj1.zip through Blackboard. The zip file should contain the programs **sorting.c**, **main.c** and the header file **sorting.h**. If the zip file contains a makefile, we will use that file to make your executable. The zip file should also contain the report. The grade depends on the correctness of your program, the efficiency of your program, the clarity of your program documentation and report. The two sorting functions will account for at least 50% and at most 70% of the entire project grade. The other functions and the report will account for the remainder of the grade 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. Memory leak will result in at least 50% penalty.

Given:
Besides the main routine, we provide sample input files (in proj1) for you to evaluate the runtimes, and numbers of comparisons and moves of your sorting algorithms. The number in the name refers to the number of long integers the file is associated with. For example, 1000.txt contains 1000 long integers and 1000s.txt should contain 1000 sorted long integers from 1000.txt, 1000seq.txt contains the sequence used for sorting. In particular, 1000s.txt and 1000seq.txt are created by proj1 by the following command:

./proj1 i 1000.txt 1000seq.txt 1000s.txt

Start sorting!