ECE368 Project #2

Due Wednesday, October 9, 11:59pm

Description

This project is to be completed on your own. Similar to Project 1, you will implement Shell sort. However, in this project, you will use only bubble sort as the basic sorting routine. The major difference here is that you will use a linked-list to store the long integers, instead of using an array for storage. Consequently, your sorting will have to be performed on a linked-list. (Although you are allowed to implement linked-lists using an array, your sorting will still have to be performed on a linked-list. You may not assume that you can access the array directly.)

For this project, you will use only one sequence for Shell sort: $h(1) = 1$, $h(i) = 2 \times h(i-1) + 1$, for $i > 1$. The sequence to be used is $\{1 = h(1), 3 = h(2), \ldots, h(p)\}$, where $h(p)$ is the largest integer that is smaller than the number of integers to be sorted. Your Shell sort implementation will have to sort $h(p)$ linked-lists, then $h(p-1)$ linked-lists, ..., and eventually 1 linked-list. In other words, the array in Project 1 is now a linked-list in this project, and each subarray in Project 1 is also a linked-list. In fact, you are not allowed to use an array at all for this project (except when you use an array implementation of linked-lists).

In this project, you will use the following user-defined type to store integers:

```
typedef struct _node {
   long value;
   struct _node *next;
} Node;
```

If you have to, you will use the following user-defined type to store a linked-list of linked-lists:

```
typedef struct _list {
   Node *node;
   struct _list *next;
} List;
```

This structure may be useful for you to maintain $h(k)$ linked-lists, $1 \leq k \leq p$, in your Shell sort implementation. There is a chance that you may not need this structure.

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 files.

```
Node *Load_File(char *Filename)
```

The file contains 1 (long) integers in each line. Note that this input file format is different from project 1. In project 1, the first line of the input file tells you the number of integers in the remainder of the file. In this project, the number of integers is not specified. The function should read all integers in the file into a linked-list and return the linked-list.

```
int Save_File(char *Filename, Node *list)
```

The function saves `list` to an external file specified by `Filename`. The output file and the input file have the same format. The returned value of this function indicates the number of integers successfully written into the file.

```
void Shell_Sort(Node *list)
```

This function takes in a `list` of long integers and sort them. To correctly apply Shell sort, you would have to know the number of elements in the `list` and find $h(p)$ accordingly.

(Alternatively, you can choose to implement `Node *Shell_Sort(Node *list)`, which returns the first node of the sorted list.)

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.c sorting_main.c -o proj2
```

When the following command is issued,

```
./proj2 input.txt output.txt
```

the program should read in `input.txt` to store the integers to be sorted into a linked-list, run Shell sort on the linked-list, 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:

```
I/O time:  AAAA
Sorting time:  BBBB
```

where `AAAA` and `BBBB`, all in `%le` format, report the statistics you have collected in your program.

Report you will have to write:

You should write a (brief) report that contains the following items:

- A tabulation of the I/O run-times and sorting run-times obtained from running your code on some sample input files. You should comment on how the I/O run-times and sorting run-times grow as the problem size increases, i.e., the time complexity of your routines.
- A comparison of the I/O run-times and sorting run-times obtained in this project and in project 1 (for Sequence 1 and Shell sort using bubble sort only). Explain any significant differences that you observe.
- The space complexity of your implementation of the Shell sort with bubble sort using a linked list. Here, you should not count the linked list used to store the input.

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

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 that all the files that have been opened are closed and all the memory that have been allocated are freed before the program exits. The final grade will also take into account the time and space efficiency of your code (with respect to your classmates' programs).

Given:
You may use similar input files as in project 1, except that you want to remove the first integer in each file. If you choose to use the exact same input files as in project 1, your program should sort all numbers in the files, including the first number in each of the input files.

Start sorting EARLY!