ECE368 Project #2

Due Friday, February 28, 11:59pm

Description

This project is to be completed on your own. Similar to Project 1, you will implement Shell sort. The major differences here are that (1) you will implement only insertion sort, and (2) 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.

For this project, you will use the same sequence as in Project 1 for Shell sort. For each *k* in the sequence, Your Shell sort implementation will have to sort *k* linked-lists. In other words, each subarray can be implemented as a linked-list.

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

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

You may use the following user-defined type to store a linked-list of linked-lists (to be exact, a linked-list of pointers to `Node`):

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

This structure is probably useful for you to maintain *k* linked-lists, where *k* is a number in your sequence.

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.

```
Node *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 the sequence accordingly. The first node of the sorted list is returned by the function.

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.

A few tips about writing your functions: Although this project does not ask you to count the numbers of data comparisons and swaps. It is recommended that you keep them initially so that you can compare your new code against your code from Project 1.

Keep in mind that it is easy to traverse a singly linked-list in only one direction. Therefore, you should think about whether you should keep the integers in a sorted linked-list in ascending order or in descending order. Moreover, you have to perform sorting many times. Do you always sort a linked-list in ascending or descending order, or do you want to sometimes sort in ascending order and sometimes in descending order?

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.
- The space complexity of your program. Do not include the space required by the linked list to store the integers. However, if you have to create a linked list of lists (or a linked list of pointers), you would have to account for the additional space for that linked list of lists (or pointers).
- A comparison of the I/O run-times and sorting run-times obtained in this project and in project 1 (for insertion sort only). Explain any significant differences that you observe.

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.

Bonus points will be given if your code has better performance than those of your classmates. Bonus points will also be given if your code has $O(1)$ space complexity (not counting the linked list of long integers that you read in from a file).

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!