ECE36800 Programming Assignment 5

Due Sunday, April 7, 2019, 11:59pm

Description

This project is to be completed on your own. You will implement the following two sorting algorithms:

- Quicksort
- Mergesort

The goal here is to have efficient implementations of the two algorithms.

Functions you will have to write and submit

You should declare these two functions in `sorting.h`.

```
void Quick_Sort(long *Array, int Size)
```

```
void Merge_Sort(long *Array, int Size)
```

These functions take in an `Array` of `long` integers and sort them. `Size` specifies the number of integers to be sorted.

You should define these two functions in `sorting.c`. You may declare and define other help functions in `sorting.c` However, these help functions should not be declared in `sorting.h`. It is best that these helper functions be declared as `static`. Do not name these help functions with a prefix of two underscores "__".

Other functions you may want to write (in `pa5.c`)

These functions are necessary for your own testing of your implementation of the two sorting algorithms. They should not be submitted.

The first two functions `Load_Into_Array` and `Save_From_Array` are needed to transfer the integers to be sorted to and from the files. You may reuse your code from Programming Assignment 2.

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

The size of the binary file whose name is stored in the `char` array pointed to by `Filename` should determine the number of `long` integers in the file. The size of the **binary** file should be a multiple of `sizeof(long)`. You should allocate sufficient memory to store all `long` integers in the file into an array and assign to `*Size` the number of integers you have in the array. The function should return the address of the memory allocated for the `long` integers.

You may assume that all input files that we will use to evaluate your code will be of the correct format.

Note that we will not give you an input file that stores more than `INT_MAX` `long` integers (see `limits.h` for `INT_MAX`). If the input file is empty, an array of size 0 should still be created and `*Size` be assigned 0.

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

The function saves `Array` to an external file specified by `Filename` in **binary 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 output file should be created.

You should also write a `main` function so that you can test and evaluate the performance of your implementations of these sorting algorithms. You should model your `main` function after the `main` function written in Programming Assignment 2. Use the option "-q" to invoke quicksort and "-m" to invoke mergesort.

Submission

The project requires the submission (electronically) of `sorting.h`, `sorting.c`. You should create a zip file called `pa5.zip` that contains `sorting.h` and `sorting.c` and submit the zip file. For example, you can create a zip file with the following command:

```
zip pa5.zip sorting.c sorting.h
```

Note that we do not require you to submit `pa5.c`. We will use our own `pa5.c` to test your code in `sorting.h` and `sorting.c`.

Grading

The grade depends on the correctness and efficiency of your program.

The quicksort function will account for 50 points and the mergesort function will account for the remaining 50 points. A function is deemed correct only if it can complete sorting within a pre-specified time limit (meeting the expected time complexity of $O(n \log n)$, where n is the number of `long` integers to be sorted). Bonus points will be awarded to submissons that are among the fastest. Up to 10 points will be awarded for the quicksort function and up to 10 points will be awarded for the mergesort function.

Given

We provide an executable to generate input files for the evaluation of your implementations. The executable is called `generate`. The executable is compiled on ecegrid. Therefore, you have to save it on ecegrid and run it on ecegrid. You may have to change the mode of the file so that you can run it. To make it an executable,

```
chmod u+x generate
```

To run the program,

```
./generate size filename
```

where `size` is an integer specifying how many `long` integers you want, and `filename` is the name of the file to store these integers. For example,

```
./generate 1000000 1M.b
```

generates a file with 1 million `long` integers and stores them in `1M.b`.

When you are working on your program, please use the following command to compile your program (assuming that you have `sorting.c`, `sorting.h`, and `sorting_main.c`). The flag `-g` allows you to debug your program.

```
gcc -g -std=c99 -Wall -Wshadow -Wvla -pedantic sorting.c pa5.c -o pa5
```

When we evaluate your submission, we will use the following command to compile your program:

```
gcc -std=c99 -Wall -Wshadow -Wvla -pedantic -O3 sorting.c pa5.c -o pa5
```

We have dropped the `-g` flag and added an optimization flag `-O3`. Note that it is upper-case letter `O` (not number zero). After you have debugged your program, you should compile your program with the optimization option and you should observe a difference in the run-time of your program.

We will call the program to perform quicksort as follows:

```
./pa5 -q input.b output.b
```

We will call the program to perform mergesort as follows:

```
./pa5 -m input.b output.b
```

Note that you do not have to submit `pa5.c`. We will provide the `pa5.c` file in our evaluation of your submission.

Getting started

Copy over the files from the Blackboard website. Check out the Blackboard webpage for any updates to these instructions. *Start sorting!*