EEE2020-03 **Spring 2017**

Assignment 4: Heap & Sorting

Due: June 12, 2017, 11:59PM

1 Installation

- Obtain the baseline code structure from YSCEC or the link below. Try using `wget` and `unzip` commands in Linux terminal to download and extract the zip file. `$` sign is typically used to indicate that it is a Linux terminal command. You do not type this sign in Linux terminal.
```
$ wget http://web.yonsei.ac.kr/icsl/teaching/assignments/sorting.zip
$ unzip sorting.zip
```
- Try compiling the baseline code by following the steps below. The baseline code may compile and run to the end, but the output will remain unsorted since sorting functions are incomplete. You will have to implement two different sorting algorithms to complete this assignment.
  - In Windows Visual Studio (or any equivalent C++ development software), create an empty project and load all header and source files into the project. Note that the `input/` directory that comes with `sorting.zip` must be placed inside your Visual Studio working directory to let it find input files. Use Ctrl+F5 or any similar options to compile the project files.
  - In Linux, type the following command to compile sources and generate an executable file.
```
$ cd sorting/
$ make
```
    The following command removes all object files (i.e., `*.o`) and the executable. It is generally a good idea to type `make clean` and then `make` especially when major changes are made to the files.
```
$ make clean
```

2 Implementation

Complete the `sorting.cc` file to make the sorting functions working. The following functions need to be implemented to complete this assignment. For each sorting function definition, `*m_array` is a pointer to the input array; i^{th} element in the array can be accessed by `m_array[i]`. `m_size` means the size of array (or the number of elements in the array).

- `void heapsort(uint32_t *m_array, const size_t m_size);`
- `void mergesort(uint32_t *m_array, const size_t m_size);`

3 Testing

`main.cc` already includes a test array to verify the correctness of sorting functions above. The array has 20 elements in random order. Find the comment line of `/* TEST YOUR CODE HERE TO SEE IF SORTING IS WORKING */`. You may modify the test array to check if your function implementations in `sorting.cc` are working fine. In Windows Visual Studio, Ctrl+F5 will compile the sources and run the program at once. In Linux, type the command below in the terminal to run the program. If changes were made to the source codes, run the `make` command before executing the program.
```
$ ./sorting
```

When the program is executed, it first asks which type of sorting function to run. Type `h` to use heapsort or `m` for mergesort.
```
Select sorting type, heapsort(h) or mergesort(m):  h
```

4 Results

After the test cases pass, the main function will proceed to validate the sorting functions with large input data set. Do not modify this part of the code in `main.cc` since the output of this process will be used for grading.

At the beginning of validation process, the program will ask your input to choose a data set to use. Type in the last digit of your student ID. For instance, if your student ID is 2016123456, then put 6 and then press enter.

```
Last digit of your student ID: 6
```

You should expect the results as follows; numbers in the examples below are randomly made up.

If heapsort is selected, then:

```
Last digit of your student ID: 6

Sorting 1000000 numbers:
Elapsed time = 235.796 msec
array[2017] after sorting = 20573

Heapsort done.
```

Or if mergesort is run, then:

```
Last digit of your student ID: 6

Sorting 1000000 numbers:
Elapsed time = 261.913 msec
array[2017] after sorting = 21946

Mergesort done.
```

5 Submission

5.1 Record the results.

Open an empty text file, and record your simulation results. For example, copy and paste 7 lines from `Sorting 1000000 numbers:` to `Heapsort/Mergesort done.` including the blank line in the middle, as shown in the examples above. Note that the results must be produced by using the correct input data set, which is determined by the last digit of your student ID that you type in at the beginning of validation process. Save the text file, and name it as `<your student ID>.txt` such as `2016123456.txt`. The file should include the results of both sorting algorithms, which mean you need to run the program twice, once for heapsort and another for mergesort.

5.2 Clean your source code directory.

Clean up your working directory by removing all object files (i.e., `*.o`), executable file (i.e., `sorting`), input directory (i.e., `inputs/`), and any other files that Windows, Mac, Linux, or you had created. If you used Mac OS, make sure that hidden directories such as `__MACOSX` is removed. In the end, only the following files should be left in the `sorting/` directory.

- `file_handler.cc`
- `file_handler.h`
- `gettimeofday.h`
- `sorting.cc`
- `sorting.h`
- `stopwatch.cc`
- `stopwatch.h`
- `main.cc`
- `Makefile`
- A result file such as `2016123456.txt`

5.3 Make your source code zip file ready.

After the directory is cleaned up, create a zip file named after your student ID such as `2016123456.zip`. You may use the following Linux command to create one.

```
$ zip -r 2016123456.zip sorting/
```

5.4 Submit your zip file through YSCEC.

The final step is to upload your zip file at YSCEC. The zip file should include all the C++ files, Makefile, and the result text file inside the `sorting/` directory.

6 Grading Rules

The following is a general guideline for grading this assignment. 20-point scale will be used this time. Grading rules may be subject to change.

- **+2 points:** Student reports an apparent bug in the baseline code structure.
- **-1 point:** Submitted zip file comes with uncleaned working directory or missing files.
- **-1 point:** Submitted zip or result file is not named after student ID.
- **-6 points:** Students may still submit assignments during 1-day graceful period with late submission penalty.
- **-16 points:** Result file is not reproducible by submitted source code.
- **-16 points:** Submitted source code is nearly identical to the baseline code.
- **-20 points:** No submission, or source code turns out to be identical to those of other classmates.

And the following rules will apply to each sorting function.

- **-1 point:** Source code has insufficient comment lines.
- **-2 points:** Source code compiles and runs without errors, but result file is incorrect or no result file is found.
- **-4 points:** Source code does not compile or run to completion.

7 Grade Dispute

Your teaching assistant (TA) will evaluate assignments. If you think your assignment score is incorrect for any reasons, discuss your concerns with the TA. In case no agreement is made between you and the TA, elevate the case to the instructor to review your assignment. Refer to the course website for the contact information of TA: `http://web.yonsei.ac.kr/icsl/teaching/eee2020.html`