Project 3: Sorter

1. Introduction

In this assignment, you will implement one module, Sorter, that sorts an array of integers in ascending order, in which the user of your Sorter module can specify which one of sorting algorithms is used, Insertion sort, Merge sort or Quicksort. After implementing your module, you will write a report about the three sorting algorithms. (See Section 4 for more details.)

The code skeleton provides you with two files (**Sorter.hpp**, **test.cpp**) and is under **/comp/15/files/p3** on the CS server.

1. Log in to the CS homework server.
2. Move to your **comp15** directory that you created in Lab 1.
3. Create a directory named **project3** under the comp15 directory.
4. Move to the project directory.
5. Copy the code skeleton to the current working directory.
6. Leverage the features of Git to manage your progress toward writing the program.

Suggestion: Consider implementing the **isSorted()** function in **test.cpp** first. (See Another requirement section for more details.)

You will at least need to create **Sorter.cpp** file by yourself. Note that the file name matters for the grading purpose and is case-sensitive. Note also that the public section of **Sorter.hpp** cannot be modified. Finally, you are expected to write your own tests in **test.cpp**.

2. Specifications

Sorter class

`Sorter();`

This sorter is initialized to be one whose sorting mode is set to be `Mode::INSERTION_SORT`.

`Sorter(Mode mode);`

This sorter is initialized to be one whose sorting mode is set to be the given "mode". The given "mode" is one of `Mode::INSERTION_SORT`, `Mode::MERGE_SORT`, or `Mode::QUICK_SORT`.

`Sorter(const Sorter& other);`
`Sorter& operator=(const Sorter& other);`
This sorter is initialized to be one that holds the same information that the given sorter holds.

`~Sorter();`
You are supposed to release (memory) resources, if necessary.

`void sort(int* const array, int size) const;`
Sort the array of integers pointed by the given "array" in ascending order using the sorting algorithm that this sorter currently holds. The given "size" is the size of the given array.
If the given "array" points to `nullptr` and/or if the given "size" is less than 1, then throw a `std::invalid_argument` exception with the message (case-sensitive):
Invalid Argument

`void sort(int* const array, int size, Mode mode) const;`
Sort the array of integers pointed by the given "array" in ascending order using the sorting algorithm corresponding to the given sorting "mode". The given "size" is the size of the given array.
If the given "array" points to `nullptr` and/or if the given "size" is less than 1, then throw a `std::invalid_argument` exception with the message (case-sensitive):
Invalid Argument
Note that calling this method does not change the sorting mode that this sorter holds.

`void set(Mode mode);`
Set the given "mode" to be the sorting mode that this sorter holds.

`Mode getMode() const;`
Return the sorting mode that this sorter holds.

Another requirement

Implement the following function in **test.cpp**, and leverage it when testing your Sorter module.

`bool isSorted(int* const array, int size);`
Return `true` if the array of integers pointed by the given "array", whose size is the given "size", is sorted in ascending order; `false`, otherwise.

3. Testing

Modify the **test.cpp** by adding your own test cases to ensure that your modules function correctly. The given **test.cpp** includes two example test cases using assert(). You are encouraged to add as many test cases as necessary to make you feel confident about your modules.

4. Writing a Report

After implementing your Sorter module and testing its functionality, try using your module to sort a variety of arrays of integers, for example, of different sizes, of different characteristics, and so on. Then, write a report that includes a discussion of each sorting algorithm in terms of time complexity and includes any of your findings.

The report should be precise and concise and should be at most one page long per sorting algorithm. The report should be submitted as **PDF**. If you plan to perform an empirical analysis of sorting algorithms, similar to what we did in the In-Class Activity 3, you may find the **/comp/15/files/a3/test.cpp** file useful. It will be a good idea to use colors and to include figures in your report.

5. README

Create the README file that includes the following categories with appropriate section headers.

1. **Name**: The programmer's name.
2. **Date**: The last updated date of the program code, including README.
3. **Summary**: A brief summary of the project. (e.g. What does the program do? How do you use the modules? etc.)
4. **Files**: A list of files that are necessary to build and test the program.
5. **Instructions**: A sequence of commands to compile and test the program. Note that you are expected to report procedures without using the make command.
6. **References**: A list of citations to information used to complete the project.
7. **Post evaluation on planning**: Did your project go smoothly, as planned? If not, which parts were difficult? What were the causes? How will you improve your planning for future assignments?

6. Submission

Submit your files listed below using Gradescope.
Files: **Sorter.hpp Sorter.cpp test.cpp README report.pdf**