Coding Test

The idea is to implement a simple Producer/Consumer pattern in Linux to sort input items following well known best practices of C++:

1. The project must be implemented in C++, and compiled using CMake.
2. The project must be uploaded to a public repository in github and the address of the repo be shared.
3. The main process must be a command line executable which loads a file as input file(passed as an argument) in which each line of the file will represent an item to process (which is a string composed by numeric characters or blank spaces, e.g: 15423123 931234 1) and an output file in which the results will be printed, as well as a third argument to choose the sorting algorithm. The amount of characters in each item will never exceed 100 chars. The amount of lines in the input file will never exceed 10000 lines.
    1. So something like ***./ProducerConsumer /path/to/InputFile /path/to/OutputFile algorithmToSort***
4. For each item, a new WorkerThread must be created which will process the item (to a maximum of 4 worker threads). Inside each thread:
    1. The thread must split each character of the item into a vector and sort them (**no sort() is allowed, an implementation of two sorting algorithms must be made manually). The algorithm use for processing is determined by the third argument when executing the program. This is one of the most important parts of the exercise. The interviewee must explain (in a README file, for example) which two algorithms chose and why.** If a blank space is found as part of the item, the thread must wait one second and not include the blank space in the vector.
    2. The thread must them print the characters of the vectors separated by commas into the output file detailed as second argument (each item in a new line, the file shared by every worker thread of the module).
5. The points to evaluate will be:
    1. Structure of the code in the github repo.
    2. Instructions to compile the code of the github repo and produce a C++ executable.
    3. Cleanliness of the code, based on cppcheck, and documentation of the code.
    4. No warning or errors in compilation.
    5. Limit the use of external libraries.
    6. Good practices related to names of the variables and methods, length of methods, lines, spaces, cyclomatic complexity, etc.
    7. Architecture of the code. We expect a clean module with the Producer Consumer, following known design patterns (specially for choosing the running algorithm to use), understanding of async programming as well as avoiding racing conditions, deadlocks, unnecessary polling, while trying to optimize **performance**.