

Assignment 4 - Array Performance

[Submit Assignment](#)

Due Oct 5 by 11:59pm **Points** 0 **Submitting** a file upload **File Types** cpp, txt, and in
Available Sep 28 at 12am - Oct 7 at 11:59pm 10 days

The purpose of this assignment is to perform an evaluation of the relative performance of three different array types in C++, raw arrays, `std::array`, and `std::vector`. A second aspect is to give you experience using the `<chrono>` library, which is used to obtain the timing information.

Assignment

Create a pair of files named `sortutils.hpp` and `sortutils.cpp`.

In `sortutils.hpp`, place the following declarations:

- `const std::size_t HOW_MANY_ELEMENTS = 250000;`
- `const std::uint8_t HOW_MANY_TIMES = 25;`
- `using SourceArray = std::array<int, HOW_MANY_ELEMENTS>;`

In the header file place the following prototypes and in the implementation file, write the implementation for the functions.

- `void initializeRawArrayFromStdArray(const std::array<int, HOW_MANY_ELEMENTS>& source, int dest[]);`
 - This function accepts an `std::array` and copies its elements into a raw array. The author of assumptions is that the dest array is the same size and the source array; assume valid arrays and matching sizes are passed in (for this class, not ever for production code).
- `void organPipeStdArray(std::array<int, HOW_MANY_ELEMENTS>& data);`
 - This function re-organizes the values in the data array to have an organ pipe structure. It is assumed that data is already sorted. Refer to the notes below on what this organization looks like.
- `void evaluateRawArray(const SourceArray& random, const SourceArray& sorted, const SourceArray& reversed, const SourceArray& organPipe, const SourceArray& rotated);`
- `void evaluateStdArray(const SourceArray& random, const SourceArray& sorted, const SourceArray& reversed, const SourceArray& organPipe, const SourceArray& rotated);`
- `void evaluateStdVector(const SourceArray& random, const SourceArray& sorted, const SourceArray& reversed, const SourceArray& organPipe, const SourceArray& rotated);`

Each of the evaluateX functions time and report how long it takes to sort each of the array types `HOW_MANY_TIMES`; look at the sample output to better understand this. Each function separately times how long it takes for each array type. You'll use the constant arrays passed in as originals, making a copy of them each time you want to sort it (`HOW_MANY_TIMES`). Time ONLY the sorting time, do not include any time spent making copies of the arrays or other overhead.

Use the C++ standard library sort algorithm, `std::sort`, found in the `<algorithm>` library. You can sort a raw array with `std::sort(data, data + arrayLength);` Where `data` is the name of the array and `arrayLength` is the number of

elements in the array.

You may write additional functions, not identified above, in `sortutils.cpp`, but do not place prototypes for them in `sortutils.hpp`. I wrote several other functions to support my code, simplifying the various `evaluateX` functions by relying on some support functions.

Write a driver program that prepares a `SourceArray` (as defined above) of int random numbers, ranging from -10,000,00 to 10,000,000. From this array, prepare the following additional arrays:

- sorted - A sorted version of the data in the random array. { 0, 1, 2, ..., n - 1 }
- reversed - A reversed version of the sorted array. { n - 1, n - 2, n - 3, ... 2, 1, 0 }
- organPipe - An organ pipe structuring of the sorted array. { 0, 1, 2, ..., n / 2, n / 2 - 1, ..., 2, 1, 0 }
- rotated - A rotated version of the sorted array { 1, 2, 3, ..., n - 2, n - 1, 0 }

With these arrays prepared, call each of the `evaluateX` functions to have them do a performance evaluation and report on the use of the array type (in the function name).

Standard Library Support

The standard library contains a number of functions you'll need to use for this assignment. I've linked to the [cppreference.com](https://en.cppreference.com) website for details on their use. This is an excellent source for information on the C++ language. It is a reference site, not a tutorial site, but it still is my "go to" spot for looking up specifics on C++.

- `std::sort` - <https://en.cppreference.com/w/cpp/algorithm/sort> (<https://en.cppreference.com/w/cpp/algorithm/sort>)
- `std::reverse` - <https://en.cppreference.com/w/cpp/algorithm/reverse> (<https://en.cppreference.com/w/cpp/algorithm/reverse>)
- `std::rotate` - <https://en.cppreference.com/w/cpp/algorithm/rotate> (<https://en.cppreference.com/w/cpp/algorithm/rotate>)

Other Items


The default stack size for a process is relatively small, a few tens of thousands of bytes. Because this program is creating very large arrays on the stack, the size of the stack needs to be increased. One of the ways to do this is to use compiler options to increase the size. Here is what you need to add to your `CMakeLists.txt` file to set the compiler option for each of MSVC and g++.

MSVC

Add this line for MSVC only: `set_target_properties(UnitTestRunner PROPERTIES LINK_FLAGS "/STACK:10000000")`

g++

Add this to the `target_compile_options` command (don't add any spaces): `-Wl,--stack,10000000`

A file named `TestPerformance.cpp` is provided which contains a couple of unit tests. The first test verifies your organ pipe function works correctly. The second test verifies a raw array is initialized correctly from a `std::array`. The file is available at this [link](#) .

Submission Notes

- Turn in the following files:
 - **main.cpp**
 - **sortutils.hpp**, **sortutils.cpp**
 - **TestPerformance.cpp** (provided for you)
 - **CMakeLists.txt**, **CMakeList.txt.in** (if you have one)
- CMakeLists.txt targets
 - Use **EXACTLY** `ArrayPerformance` as the name for the `add_executable` target for the application.
 - Use **EXACTLY** `UnitTestRunner` as the name for the `add_executable` target for the unit tests.
- Your code must compile without any warnings or compiler errors. See syllabus regarding code that has compiler errors.
- Your code must adhere to the CS 3460 coding standard: [link](#)
- Your code must be formatted through the use of the following clang-format configuration file: [link](#)

Sample Run

The following is from a run on my desktop computer in my USU office, using a "Release" build from Visual Studio. Your output should follow this format; of course, your results will differ.

```
--- Raw Array Performance ---
Random Time      : 471 ms
Sorted Time      : 60 ms
Reversed Time    : 69 ms
Organ Pipe Time  : 199 ms
Rotated Time     : 68 ms

--- std::array Performance ---
Random Time      : 476 ms
Sorted Time      : 55 ms
Reversed Time    : 68 ms
Organ Pipe Time  : 172 ms
Rotated Time     : 63 ms

--- std::vector Performance ---
Random Time      : 449 ms
Sorted Time      : 53 ms
Reversed Time    : 60 ms
Organ Pipe Time  : 175 ms
Rotated Time     : 57 ms
```