Sorting Algorithms Visualization

Learning Objectives

- Implement several sorting algorithms in C.
- Visualize their behavior using the SDL2 library.
- Measure and compare performance (memory accesses, comparisons, execution time).
- Work with various data types : integers, floats, and user-defined structures.
- Discover generic programming through comparison functions passed as pointers.

Project Organization

The project is progressive : each step adds new features. More advanced students are encouraged to complete the extensions.

Step 1 : Project Setup

1. Create a C program using **SDL2** that displays an array of integers as vertical bars.
   - Window size : 800×600 (modifiable).
   - Generate an array of N integers (e.g., $N = 100$).
   - Each integer is represented by a bar whose height corresponds to its value.
2. Implement a **step-by-step Bubble Sort visualization** :
   - On each comparison or swap, refresh the display.
   - Highlight the compared elements with a different color.
   - Use a small delay (`SDL_Delay`) to make the animation visible.

Step 2 : Additional Sorting Algorithms

Implement several other sorting algorithms and allow the user to select them via keyboard :

- Selection Sort
- Insertion Sort
- QuickSort
- MergeSort

All algorithms must be executed step by step to allow visualization.

Step 3 : Performance Measurement

Add a measurement system :

- Count of memory accesses (reads and writes).
- Count of comparisons.
- Total execution time.

Statistics must be displayed in real time inside the SDL2 window.

Step 4 : Input Distributions

Add different modes to generate the initial array :

- Uniform random
- Already sorted
- Reverse sorted
- Nearly sorted (small disorder)
- Special distributions (e.g., many duplicates, pyramid shape)

This highlights how some algorithms behave very differently depending on the input.

Step 5 : Advanced Data Types

Extend your sorting implementations :

1. **Floats :** Replace integers with floating-point numbers.
2. **Structures :** Sort an array of structures, for example :

```
typedef struct {
    char name[20];
    int age;
    float grade;
} Student;
```

   Sort students by age, grade, etc.

3. **Generic comparison functions :** Implement your algorithms so that they use a function pointer for comparison :

```
typedef int (*compare_func)(const void*, const void*);
```

   This makes the algorithms generic, similar to the standard C function `qsort`.

Step 6 : Extensions

Optional improvements for advanced students :

- Compare algorithms visually by running several sorts in parallel in different areas of the window.
- Control the animation speed with the keyboard.
- Pause/resume the animation.
- Allow the user to change the array size dynamically.
- Export statistics to a CSV file for external analysis.

Recommended File Organization

- `main.c` : SDL loop and program entry point.
- `sorting.c / sorting.h` : sorting algorithms (instrumented versions).
- `visual.c / visual.h` : SDL2 drawing functions.
- `stats.c / stats.h` : performance counters and measurements.
- `utils.c / utils.h` : array generation and helper functions.

> **Warning.** All of your code must be written in English (function names, variables, comments, etc.).

For submission, upload a compressed folder (ZIP) named `ProjectC.zip` to Moodle. Late submission is not allowed. Make sure to upload regular versions to avoid last-minute issues.

Evaluation

Your code must compile with `gcc` (version 11.4.0) on Ubuntu 22.04.2 with the flag :

- `-Werror`

We strongly encourage you to use a virtual machine if you are developing on Windows, to ensure your code compiles. See for guidance.

Useful Links

- Create a Virtual Machine
- Download Ubuntu
- GCC Compiler Documentation
- Visual Studio
- Github
- Github with Visual Studio
- Good C Programming Habits
- C Programming Rules
- Visual example of sorting algorithms
- WSL with Visual Studio Code