
浙江工业大学

数据结构课程设计

2024/2025(1)



课设题目 排序算法可视化系统

学生姓名 郑华展、任翔

学生学号

学生班级 软工06

任课教师 毛国红

提交日期 2025.01

计算机科学与技术学院

目录

一、实验内容及要求.....	4
1.1 问题描述	4
1.2 基本要求	4
1.3 实现提示	4
1.4 运行结果要求	5
二、实验开发环境.....	5
三、实验课题分析.....	5
3.1 系统总体设计	5
3.2 系统功能设计	6
3.3 类的设计	7
四、调试分析.....	11
4.1 实验的调试和测试	11
4.2 技术难点分析	12
4.3 调试测试错误分析	13
五、测试结果分析.....	14
5.1 测试结果展示	14
5.2 收获与不足	18
六、附录：源代码.....	18

图目录

Pic 1系统信息	5
Pic 2系统模块结构图	6
Pic 3Sort基类结构设计图	7
Pic 4主程序的设计流程图	11
Pic 5手动模式示例图	12
Pic 6自动模式示例图	12
Pic 7迭代容器元素的不同方法	13
Pic 8基于多线程的排序任务控制代码示例	14
Pic 9界面窗口图	15
Pic 10自动模式下Random数据测试示例图	15
Pic 11 自动模式下Sorted数据测试示例图	16
Pic 12自动模式下Reversed数据测试示例图	16
Pic 13自动模式下部分排序数据测试示例图	17
Pic 14自动模式下重复数据测试示例图	17
Pic 15自动模式下重复数据测试示例图	18

表目录

Sheet 1 Sort 类的成员函数	7
Sheet 2 Machine 类的成员函数及部分成员变量	8
Sheet 3 Machine 类的成员函数及部分成员变量	9
Sheet 4 Windows 类的成员函数及部分成员变量	9

数据结构课程设计(排序算法可视化系统)实验报告

一、实验内容及要求

1.1 问题描述

设计一个程序来可视化各种排序算法的执行过程。这个程序应该能够生成一组随机的数字，然后使用多种排序算法（如冒泡排序、插入排序、选择排序、快速排序、归并排序等）对它们进行排序。在排序过程中，程序应该在控制台上以动态的方式显示排序的每一步，以便用户可以看到排序算法是如何一步步地将数字排序的。同时，程序应该提供一种方式，让用户可以选择不同的排序算法，并可以控制排序过程的播放速度。例如，用户可以选择以慢速或快速的方式播放排序过程

1.2 基本要求

- （1）自己编程实现多种排序算法，不允许使用标准模板类的排序函数。测试时，各种情况都需要测试，并附上测试截图。要求采用类的设计思路，不允许出现类以外的函数定义，但允许友元函数。
- （2）设计图形用户界面，可以清楚地显示排序过程中数组的状态变化。用户应该能够控制排序过程的播放速度
- （3）实现随机数生成功能，在不需要进行可视化时，要求排序数据量大于 10000。并可输出排序信息，包括运行时间、比较次数、交换次数、内存使用量、稳定性、函数调用次数、循环次数和嵌套深度信息，并且可以按照不同的指标对排序算法的优劣进行排名。
- （4）要求采用多文件方式：.h 文件存储类的声明，.cpp 文件存储类的实现，主函数 main 存储在另外一个单独的 cpp 文件中。如果采用类模板，则类的声明和实现都放在.h 文件中。
- （5）要求测试例子要比较详尽，各种极限情况也要考虑到，测试的输出信息要详细易懂，表明各个功能的执行正确

1.3 实现提示

- （1）考虑到控制台单屏输出内容有限，当选择进行排序可视化展示时，可以将排序数据控制在 20 个以内。
- （2）可以使用数组或向量来存储要排序的数字。在进行排序时，需要在每一步都更新显示，以反映数组的当前状态；
- （3）在控制台上显示排序过程可能会比较复杂，同学们可以自行灵活设计。一种可行的方法是在每一行上打印出一个表示数字大小的条形图。例如，如果一个数字的值是 10，那么就打印出 10 个星号或其他字符。每一行代表一个数字，每次输出当前排序步骤的结果后，清屏再输出下一步骤结果

(4) 对于控制排序过程的播放速度，一种可能的方法是在每一步排序后都暂停一定的时间。这个时间可以由用户通过输入来调整。

1.4 运行结果要求

要求有程序菜单，能够用类的方式实现课堂教学的基本排序算法、随机数生成功能、排序算法可视化动画，实验报告要求有详细的功能测试截图

二、实验开发环境



```
> neofetch
      -\
      .0+\
      `ooo/
      `+oooo:
      `+oooooo:
      -+oooooo+:
      `/:-:++oooo+:
      `/++++/+++++++:
      `/+++++////+++++:/
      `/+000000000000000/`
      ./000SSSSS0++0SSSSSS0+`
      .00SSSSS0-`-`-`-/0SSSSS+`
      -0SSSSSS0.      :SSSSSS0.
      :0SSSSSSS/      0SSSS0++ .
      /0SSSSSSS/      +SSSS000/-
      `0SSSSS0+/-      -:/+0SSSS0+-
      `+SS0+:-`      `.-/+0S0:
      `++:.      `.-/+//
      .`      `./`

zerohzzzz@archlinux
OS: Arch Linux x86_64
Host: 20KNA01WCD ThinkPad E480
Kernel: 6.12.4-arch1-1
Uptime: 2 days, 16 hours, 8 mins
Packages: 827 (pacman)
Shell: zsh 5.9
Resolution: 1920x1080
DE: Hyprland
Theme: Catppuccin-Mocha [GTK2], Tokyo-Night [GTK3]
Icons: Tela-circle-dracula [GTK2], Tela-circle-purple [GTK3]
Terminal: kitty
CPU: Intel i5-8250U (8) @ 3.400GHz
GPU: Intel UHD Graphics 620
Memory: 2102MiB / 15893MiB
```

Pic 1系统信息

三、实验课题分析

3.1 系统总体设计

3.1.1 系统整体的功能分析

排序算法的控制台可视化程序的主要功能为：实现多种排序算法（冒泡排序、插入排序、选择排序、快速排序、归并排序等）排序过程的可视化。程序具备两种模式：手动模式和自动模式。手动模式下，用户可以通过在界面数组输入框中输入需要排序的数组（考虑到控制台单屏输出内容有限，排序数据控制在 20 个以内），选择排序算法，并在设置栏设置排序过程的播放速度、升序/降序、开启/关闭可视化等功能，就可以在输出框中看到排序过程的输出和排序算法的各种评估指标。自动模式下，用户可以选择测试数据的类型（Random、升序/降序等）来模拟极端情况，并看到排序算法完成排序所需要的时间以及其他指标。程序结束后，用户可以通过点击Exit或者使用Esc键退出程序。

3.1.2 类设计情况

Sort类：排序类，是各种排序算法的基类。

Factory类：模板化的工厂类，用于动态创建不同类型的排序算法实例。

Machine类：测试机类，用于测试排序算法。
Windows类：UI类，实现用户端展示的UI界面。

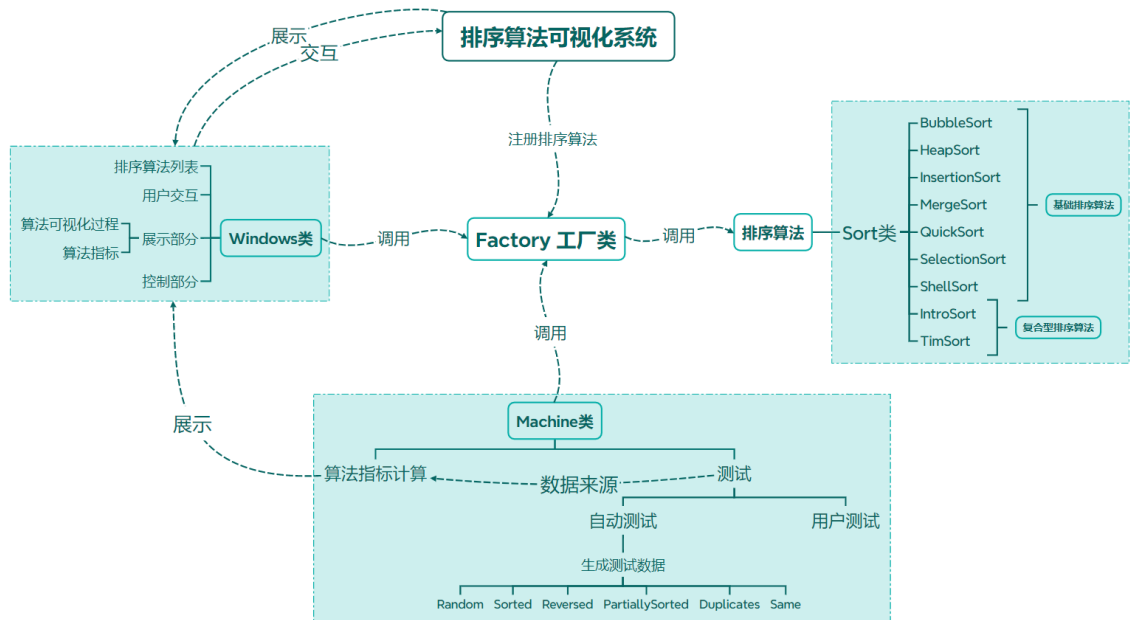
3.1.3系统结构：

Windows 界面模块：使用第三方库Ftxui实现，作为用户与系统交互的核心界面，主要实现排序算法选择、数据源设置、参数配置、动态排序过程展示及性能结果展示。

Factory 工厂模块：负责排序算法的注册和实例化管理，确保系统易于扩展和维护。

Sort 排序算法模块：实现所有排序算法的逻辑，包括冒泡排序、快速排序等，提供排序算法的标准接口。

Machine 数据与测试模块：实现数据生成、算法运行测试及性能统计功能，支持多种数据源类型和性能对比分析。



Pic 2系统模块结构图

3.2 系统功能设计

1、main()函数系统功能调用模块

main() 函数通过Factory类提供的接口注册排序算法类，并调用 Windows 类提供的 Run 方法来启动程序。因此，main() 函数仅负责程序的启动和初始化工作，而不涉及其他业务逻辑。这种设计有助于实现系统的解耦，使得各个模块能够更加独立、灵活地进行开发和维护。

2、排序模块

排序模块是整个排序算法可视化系统的核心部分，主要负责实现各种排序算法的逻辑功能，同时提供性能指标的计算和规范化的排序执行流程。模块以 **Sort基类** 为核心，通过面向对象设计原则，为不同排序算法提供统一接口和功能扩展机制。性能指标由 Sort基类 统一管理，并在 executeSort 的规范化流程中被调用。

3、可视化模块

可视化模块是排序算法可视化系统的关键组成部分，主要用于用户交互、动态展示排序算法的执行过程和性能指标。通过直观的图形化界面，用户可以较为直观的实现系统的操作，并实时观察排序算法在处理数据时的变化，从而深入理解其工作原理和效率。

4、测试模块

测试模块的主要功能是读取排序过程中的性能指标，并将其传递给 **Windows** 类。该模块分为自动测试和用户测试两部分，旨在全面覆盖系统的功能和性能需求，同时为用户提供灵活的测试配置选项。

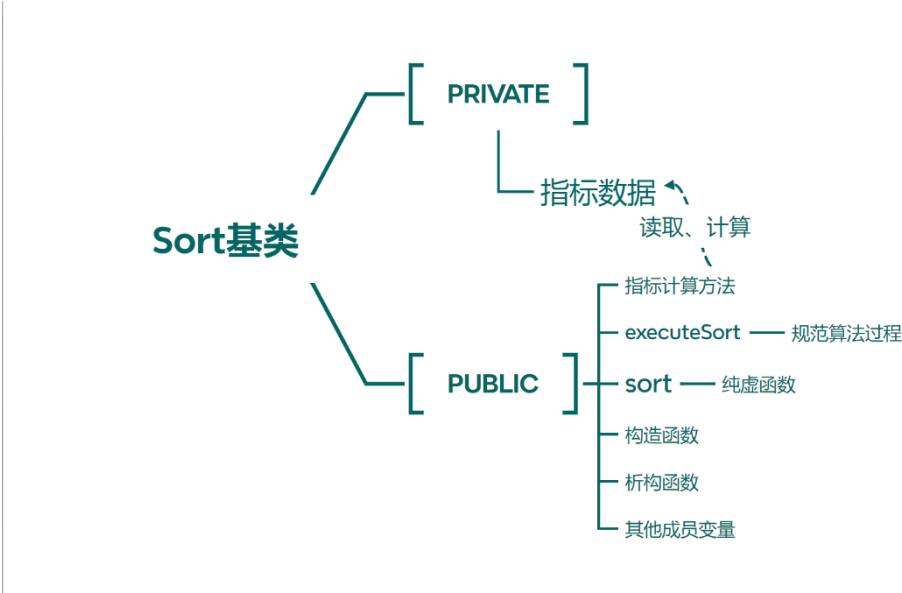
在自动测试模式下，用户可以选择生成多种类型的数据集，用以模拟不同的极端场景，对排序算法的稳定性和性能进行全面测试。在手动测试模式下，测试模块从 **Windows** 类接收用户输入的数据，并通过正则表达式解析不同的分隔符格式，显著提升数据输入的灵活性和使用便利性。此外，模块还提供清晰的测试结果输出，便于用户对算法性能进行分析和优化。

3.3 类的设计

3.3.1 类的成员函数与成员

共有4个类，Sort类，Machine类，Factory类和Windows类

1、Sort基类



Pic 3Sort基类结构设计图

Sort 是一个模板基类，用于实现各种排序算法的通用功能和性能指标统计。通过模板参数支持不同的数据类型，提供了排序算法的统一接口和运行时性能监控功能。它被设计成可以扩展以支持不同的排序算法，每个具体的排序算法将通过继承此基类并实现其纯虚函数 `sort()` 来定义。

Sheet 1 Sort 类的成员函数

类名	函数	描述
Node类	Sort(std::vector<T>& input, ftxui::ScreenInteractive& screen, size_t speed, bool GUI, int order = 0)	构造函数，初始化排序对象

	sort()	纯虚函数，子类需要实现具体的排序逻辑
	swap(T& a, T& b)	交换元素
	calculateMemoryUsage() const	计算数组占用的内存
	setSortOrder(SortOrder order)	设置排序方向
	executeSort(int speed = 0, bool gui = false)	执行排序
	updateMetrics(std::map<std::string, std::string>&)	更新指标
	resetMetrics(std::map<std::string, std::string>&)	重置所有指标

2、Machine类

Machine类是为排序算法测试和可视化设计的一个工具类。它提供了生成不同类型的待排序数组的能力，以及执行自动化和手动测试的方法。此外，Machine类还支持通过图形用户界面（GUI）进行交互，允许用户直观地观察排序过程。

Sheet 2 Machine 类的成员函数及部分成员变量

类名	函数/变量	描述
Machine类	std::unordered_map<std::string, void (Machine::*)(std::vector<int>&)>	映射字符串到成员函数指针，用于生成不同类型的测试数据
	void AutoTest(ftxui::ScreenInteractive&, int);	自动化测试函数，接收一个ScreenInteractive对象和一个整数size作为参数，用于指定要生成的数组大小
	void ManualTest(ftxui::ScreenInteractive& screen);	手动测试函数，接收一个ScreenInteractive对象作为参数，可能用于用户交互式地设置测试条件
	Random等	生成测试数据

3、Factory类

SortFactory是一个模板化的工厂类，旨在为排序算法提供一种灵活的创建机制。它允许在运行时动态选择和实例化不同类型的排序算法，以便直观地展示排序过程。此外，SortFactory还提供了注册新排序算法的能力，使得扩展和维护变得简单。

表 3-3

Sheet 3 Machine 类的成员函数及部分成员变量

类名	函数/变量	描述
Factory类	<pre>static std::unique_ptr<Sort<T>> create(const std::string&, std::vector<T>, ftxui::ScreenInteractive&, size_t, bool, int);</pre>	动态创建指定类型的排序算法实例。根据提供的类型名称type查找已注册的构造逻辑，并返回一个指向新创建的排序算法实例的智能指针。如果未找到匹配的类型，则抛出std::runtime_error异常。
	<pre>static void registerType(const std::string& name, std::function<std::unique_ptr<Sort< T>>(std::vector<T>&, ftxui::ScreenInteractive& , size_t, bool, int)> factory);</pre>	注册新的排序算法类型及其构造逻辑。将排序算法类型名称与构造函数绑定，以便后续可以通过create方法创建该类型的实例。
	<pre>static std::map<std::string, std::function<std::unique_ptr<Sort<T>>(std::v ector<T>&, ftxui::ScreenInteractive&, size_t, bool, int)>> factoryMap;</pre>	保存所有已注册的排序算法类型名称及其对应的构造逻辑。这是一个静态成员变量，意味着所有SortFactory<T>的实例共享同一个映射。这保证了即使在不同的上下文中，也可以一致地访问和创建排序算法。

4、Windows类

Windows类是专门为管理和显示排序算法的可视化界面设计的。它集成了图形用户界面（GUI）和排序算法的执行，允许用户通过交互式的方式选择不同的排序算法、配置参数，并实时观察排序过程。此外，Windows类还提供了详细的度量信息，帮助用户了解排序算法的性能特征。

Sheet 4 Windows 类的成员函数及部分成员变量

类名	函数/变量	描述
Windows类	ftxui::ScreenInteractive screen	用于管理和显示排序算法的可视化界面，采用全屏模式
	Void Show()	显示主界面，负责设置和更新GUI组件，以及处理用户交互和排序过程的可视化。

3.3.2 主要函数的设计

1、TimSort 排序算法设计：

TimSort是我们实现的复合型排序算法。结合了归并排序和插入排序的优点。TimSort 基于自然归并的思想，利用数据的局部有序性来提高效率，从而减少在最坏情况下的时间复杂度。我们使用插入

排序来处理小规模子序列，而使用归并排序来合并这些子序列。和C++自带的标准模板排序IntroSort不同的是，TimSort是稳定的算法，因此他在最坏情况下的表现比IntroSort更好。

输入数组首先被分割成多个小块（通常大小为 32）。如果一个子序列的大小超过预设阈值，就使用插入排序对该子序列进行排序。如果子序列已经有序，则不做处理。对于比较小的块，插入排序比其他算法（如归并排序）更有效。然后将已排序的小块按照归并排序的方式进行合并。通过动态选择合适的块进行合并，利用一个栈来存储当前已排序的小块，按需要进行合并。

合并这个地方很有说法。**条件1**：如果栈顶的子块与栈中倒数第二个子块的大小关系不满足某个阈值条件（通常是子块的大小比值），则不进行合并。例如，如果栈顶的子块比倒数第二个子块小很多，就不会立即合并，而是等待更多的子块入栈，以避免过早合并导致不必要的性能开销。**条件2**：如果栈中最上面三个子块的大小关系也不满足某种阈值时，TimSort 会暂停合并。具体来说，当栈中的第三个子块相对于栈顶的两个子块的大小不符合预设的合并规则时，合并操作会推迟。这是为了避免栈中的两个大块先合并，然后合并剩下的小块，造成过度合并。**条件3**：如果连续两次合并操作会导致栈中子块的大小关系变得不平衡（即相邻的子块大小差距过大），TimSort 会推迟合并。这确保了合并时不会过度压缩栈中的子块，也有助于平衡栈的深度，避免不必要的合并。只有满足这三个条件，TimSort才会进行合并。

2、IntroSort算法设计

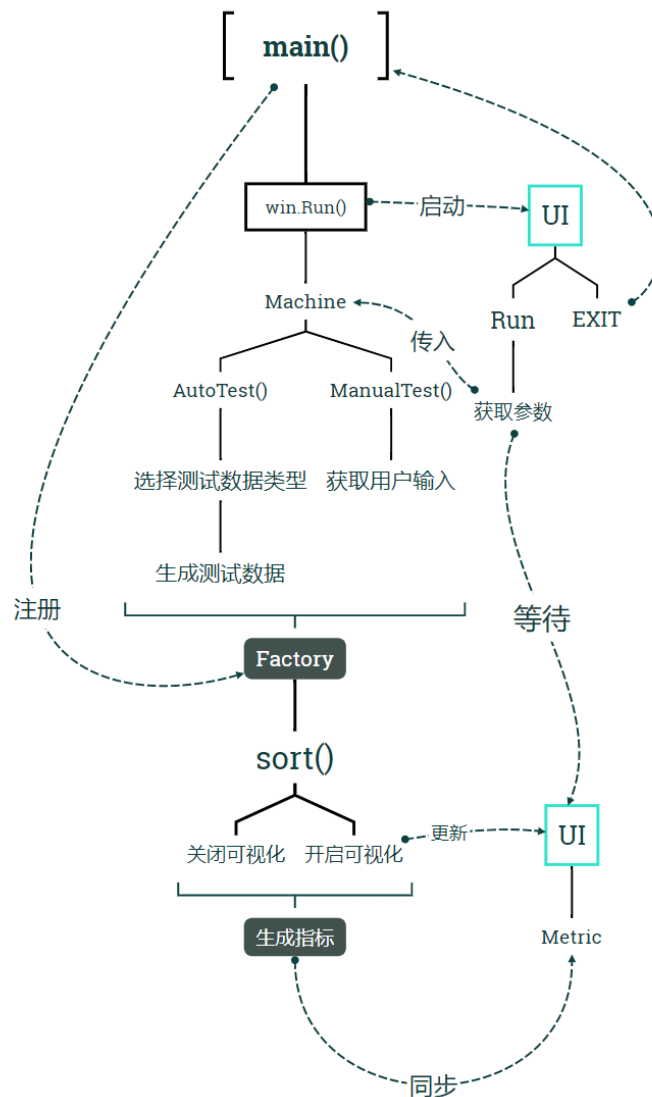
IntroSort 的核心思想是，当快速排序的递归深度超过某个阈值时，算法会切换到堆排序，保证最坏情况的时间复杂度为 $O(n \log n)$ 。同时，对于小规模子数组，IntroSort 会使用插入排序，这可以在小规模数据上提高效率。

初始阶段，IntroSort 使用快速排序（QuickSort）。快速排序基于分治法，每次选择一个基准元素（pivot），将数组分成两部分，然后递归地对这两部分进行排序。快速排序在大多数情况下非常高效，具有平均时间复杂度 $O(n \log n)$ ，但是在最坏情况下，特别是当数据已经基本有序时，可能退化为 $O(n^2)$ 。为了避免快速排序在最坏情况下退化为 $O(n^2)$ ，IntroSort 引入了一个递归深度的限制。递归深度取决于数组的大小。如果递归深度超过了这个阈值，IntroSort 会切换到堆排序。该阈值通常设置为 $2 * \log(n)$ ，这是基于树形递归结构的深度估计。

当递归深度过深时，IntroSort 会切换到堆排序。堆排序的时间复杂度是 $O(n \log n)$ ，即使是最坏情况下也能保持这个性能。堆排序通过构建一个堆（通常是最大堆），然后逐步将堆顶元素移到数组的末尾，最终得到有序数组。它的优势在于其时间复杂度是 $O(n \log n)$ ，不会退化为 $O(n^2)$ 。

在排序过程中，对于非常小的子数组，IntroSort 会使用插入排序。插入排序在处理小规模数据时具有非常低的常数因子，比其他排序算法更高效（例如，快速排序或归并排序对于小数组的常数因子较高）。一般来说，当数组的大小小于某个阈值时，IntroSort 会切换到插入排序。

3.4 主程序的设计



Pic 4主程序的设计流程图

四、调试分析

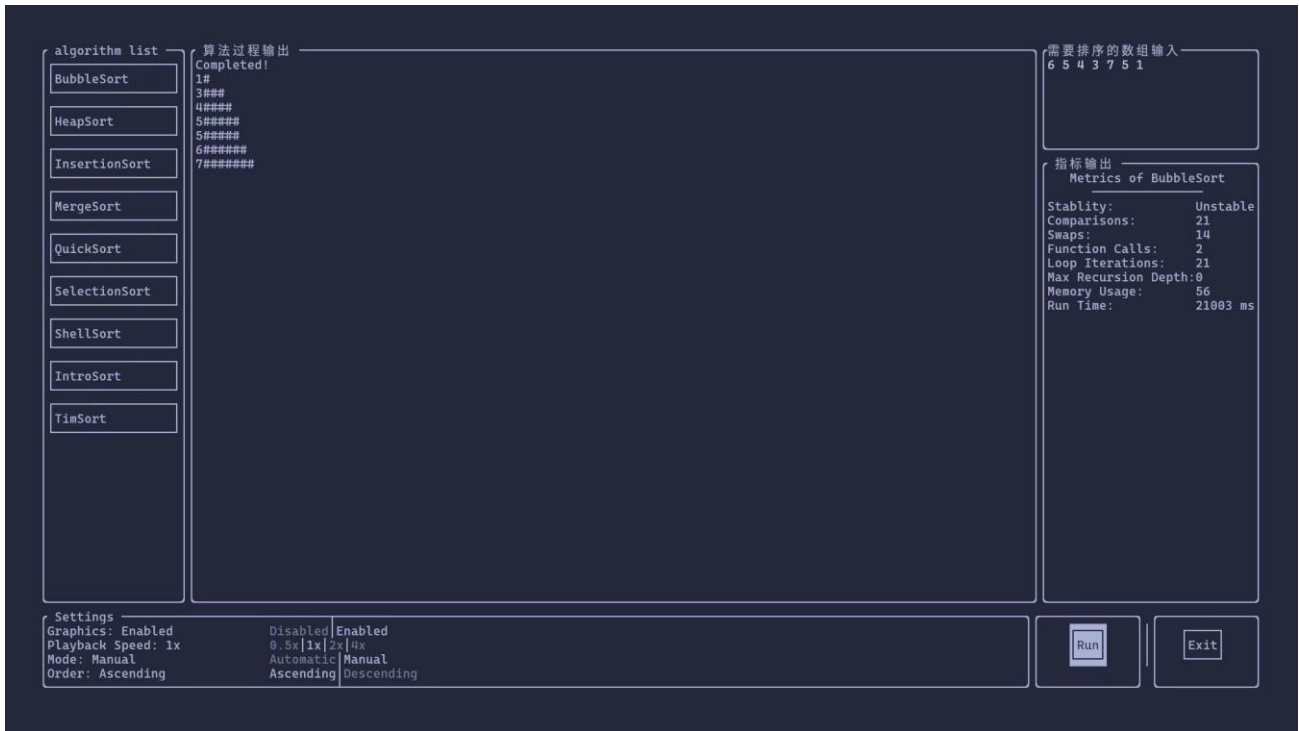
4.1 实验的调试和测试

1、输入的形式：

用户可以在Manual测试模式下从键盘输入需要排序的数组。输入的数组可以用空格、中文/英文逗号等分隔符进行分隔，程序会使用正则自动匹配输入的数字

2、手动模式调试：

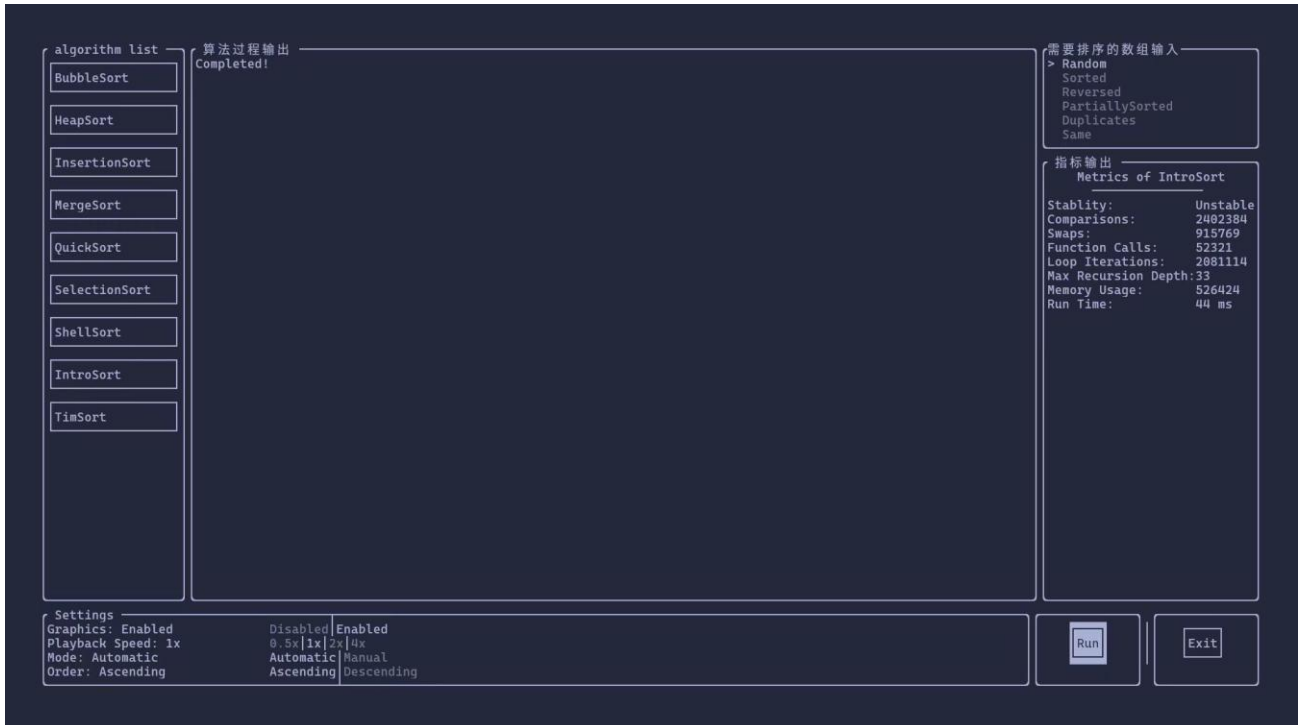
用户在数据输入框中输入需要排序的数组，选择排序算法后并在Settings栏修改控制参数，点击Run启动排序程序，可以在算法过程输出框中看到排序中每一步数组的变化。



Pic 5手动模式示例图

3、自动模式调试

用户可以在Data框中选择测试数据的类型，选择排序算法后，点击Run运行即可，等待排序结束后显示“Complete! ”，即可在指标输出框体中查看排序算法的测试指标。



Pic 6自动模式示例图

4.2 技术难点分析

1、交互性和用户界面

挑战:

如何设计用户交互界面，既能够展示排序过程，又能允许用户控制（例如调整速度、暂停、选择排序算法等）。如果用户的交互方式设计不直观，可能会影响使用体验。

解决思路：

(1) **控制面板**：提供一个简洁的控制面板，允许用户选择不同的排序算法、调整排序速度、切换可视化模式（例如条形图、数字数组等）。

(2) **实时数据调整**：允许用户动态调整输入数据（例如随机生成数据、修改数据的规模等），并观察排序算法如何适应不同的输入。

2、排序各项指标的计算

比较难通过外部去获取排序的指标，因为设计排序过程的比较、递归深度等。因此我们还是采用插桩技术，通过在算法中加入计数器，记录各项指标相关的操作。内存使用方面我们刚开始打算是给Sort排序单独开一个线程，然后再去统计线程的内存使用量和所用时间，然后发现貌似并不现实而且好像也并不符合题意，最终还是采用递归深度去计算。

4.3 调试测试错误分析

1、迭代容器元素时不同实现方式的细节导致内存泄漏

我们首先来看两种迭代方式。



Pic 7迭代容器元素的不同方法

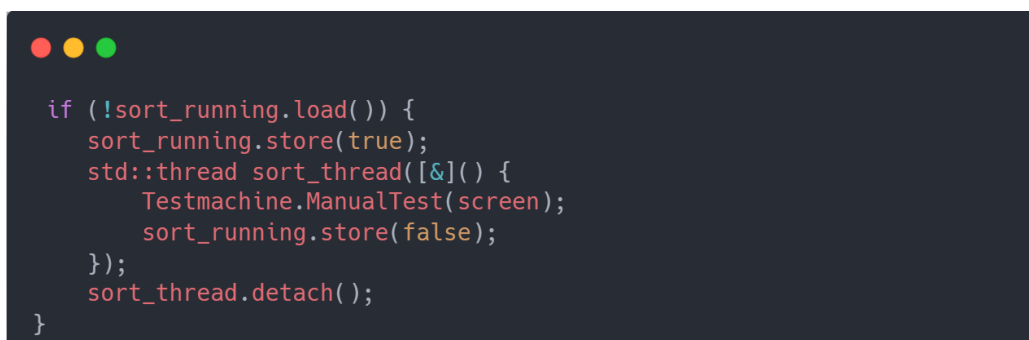
第一种写法通过显式的迭代器 (`auto it = algorithms.begin()`) 遍历 `algorithms` 容器，这种方式是 C++ 传统的迭代方式。你通过 `*it` 访问容器中的元素。第二种写法使用的是 C++11 引入的 **范围-based for loop**，它更简洁并且通常更安全。通过 `const auto& name`，你直接引用容器中的元素，而不需要显式地解引用迭代器。

这两种写法看起来没有区别，但是在第一种写法中，Lambda 捕获 `[&]` 来捕获外部变量，并且捕获的是迭代器 `it`。此时，`*it` 在 Lambda 内部是一个对 `algorithms` 容器元素的引用。如果在某些情况下，`algorithm_list_container->Add(Button(*it, ...))` 过程中，`it` 或者 `*it` 被复制或传递给其他地方，但没有适当地

管理生命周期，就可能会导致悬挂引用（dangling reference）或者内存泄漏。而第二种写法中使用 `const auto& name` 捕获的是 `name` 这个变量的引用，它直接引用了 `algorithms` 容器中的元素，而不涉及迭代器本身。引用生命周期通常比迭代器本身长，因此更安全，避免了潜在的悬挂引用问题。

2、在排序过程中，UI未能持续更新，导致播放动画无法正常进行，直到排序操作结束后才开始正常显示。

这是一个持续性的过程，难以用图片表现。很明显，我们使用的UI库是Ftxui，他这个库是基于单线程设计的，因此，Sort的过程会导致UI渲染被阻塞，直到Sort排序过程结束。我们的解决方法也很简单，我们给Sort单独去开一个线程，这样就不会阻塞主线程，UI可以继续更新和显示动画，而且这样也能更好的去管理Sort使用的内存。实现如下：



```
if (!sort_running.load()) {
    sort_running.store(true);
    std::thread sort_thread([&]() {
        Testmachine.ManualTest(screen);
        sort_running.store(false);
    });
    sort_thread.detach();
}
```

Pic 8基于多线程的排序任务控制代码示例

上图中的 `sort_running` 是一个原子布尔值变量，用来跟踪排序操作是否正在运行。其主要作用是防止在排序操作正在进行时重复启动另一个排序任务。在启动排序操作之前，使用 `store(true)` 将 `sort_running` 设置为 `true`，表示排序操作正在进行中。这样，后续代码可以检查这个值，避免再次启动排序操作，保证线程安全。而 `std::thread::detach()` 用于将线程与主线程分离，使其在后台独立运行。这样，主线程不需要等待该线程结束，也不需要显式地调用 `join()` 来等待线程完成。

五、测试结果分析

5.1 测试结果展示



Pic 9界面窗口图



Pic 10自动模式下Random数据测试示例图



Pic 11 自动模式下Sorted数据测试示例图



Pic 12 自动模式下Reversed数据测试示例图



Pic 13自动模式下部分排序数据测试示例图



Pic 14自动模式下重复数据测试示例图



Pic 15自动模式下重复数据测试示例图

5.2 收获与不足

程序使用的是第三方库Ftxui实现的可视化界面，为用户提供良好的交互环境。但是由于这个库底层设计的一些缺陷，我们的Windows类设计起来相对就比较丑。我的本意是将使用Windows去做一个Layout和Component的分离，并提供方法去注册组件和设置布局。但是由于其本身组件的实现有些缺陷，我们这种设想并没有能成功，希望后续我们能帮助Ftxui作者修复这个问题，或者通过其他的方法去实现可视化。

在类的设计上，我尽量采用了模板类的方式，以提高代码的通用性和扩展性。然而，由于时间和精力限制，目前仅仅预留了相关接口，尚未真正实现通过用户输入来判断数据类型并动态注册不同的排序类。这一点与类的设计初衷有所偏离，确实令人感到些许遗憾。不过，未来如果有时间，我会继续完善这一功能，力求为这个课程设计画上一个更加完整而圆满的句号。

六、附录：源代码