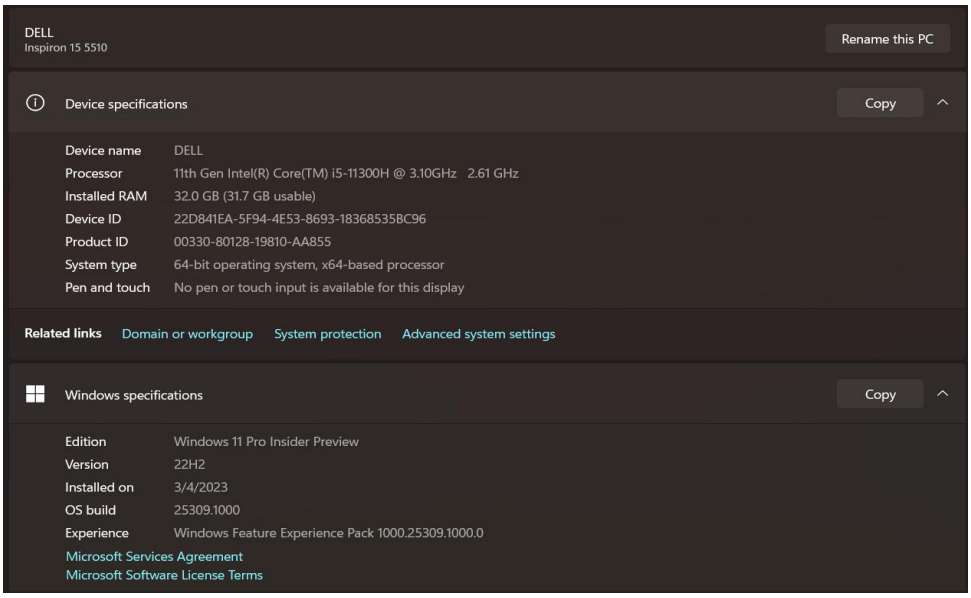
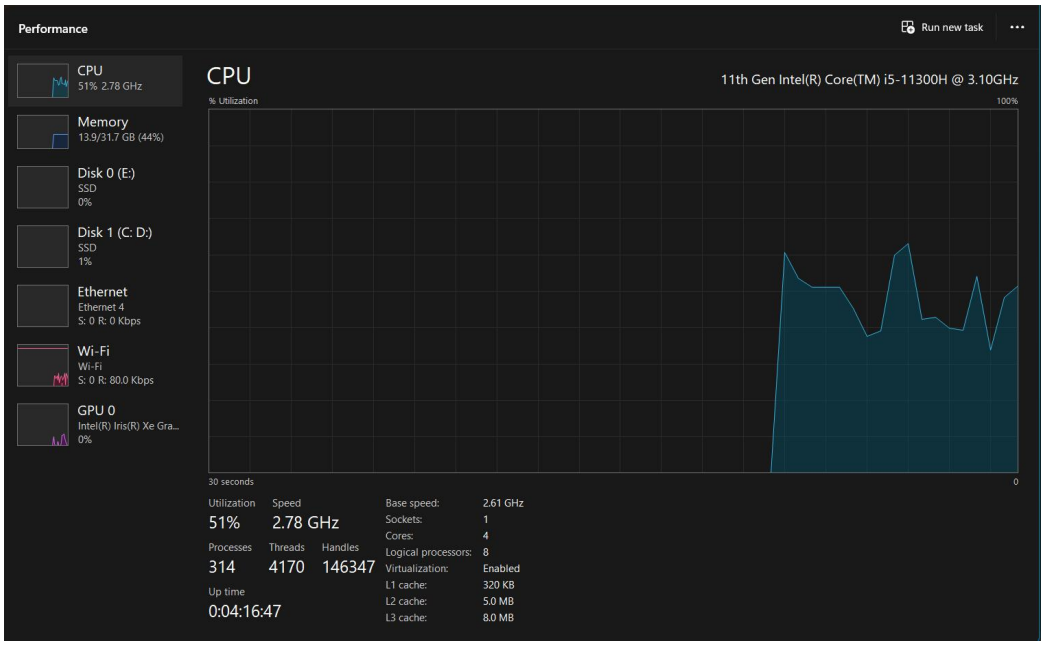
实验报告

一、 实验环境

实验中使用 x86_64 架构，DELL INSPIRON 15 5510 笔记本，配备第十一代 Intel Core i5-11300H 四核八线程处理器，标准运行频率 3.1GHz，32GB MEM，512 GB + 1 TB SSD。在 Windows 11 PRO（22H2）Dev 25314 版本系统下，运行 Microsoft Visual Studio 2022 Professional 版本 IDE 进行实验，使用 Microsoft .NET Framework 版本为 4.7.2。





二、 实验内容及分析

2.1 问题分析

排序问题：一个文本(recorder.txt)记录了大量商品价格，商品价格范围0.00~999999.99元，任选2种排序算法对商品价格按由小到大顺序排序，要求有对比测试和分析讨论的相关结果。

其它说明：（1）recorder.txt中，每个价格后跟一个空格，价格没有重复，数目可能超过1000万个；（2）recorder.txt文件下载：链接：https://pan.baidu.com/s/1rigY2CRoSJ4R0QeLFqEEiw 密码：fu1h

在课程网站按规定时间提交：说明文档（*.doc），程序源码，生成的exe文件。（注：数据文件recorder.txt不需提交）

在本次实验中，我使用 C# WPF 和 XAML 语言完成了任务。具体而言，我通过使用 XAML 描述文件来构建前端页面，然后使用 C# 编写后端代码以实现相应的功能。在某种程度上实现了前后端编程的独立性。

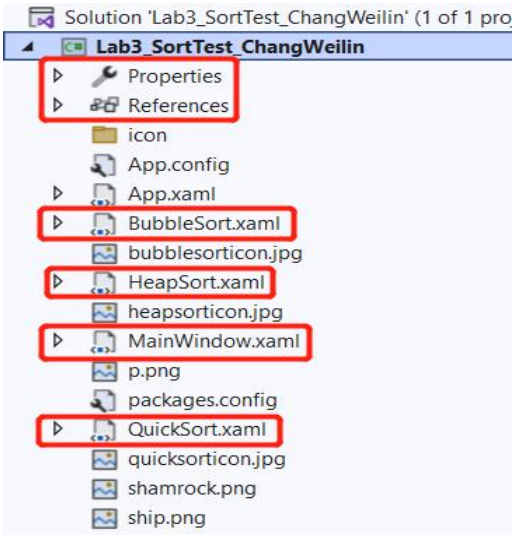
XAML（可扩展应用程序标记语言）是一种用于描述 WPF 和其他 .NET 框架的应用程序用户界面的 XML 语言。它是一种声明性语言，使得开发人员可以通过声明性的方式来定义应用程序的用户界面和行为。相较于使用代码编写用户界面，使用 XAML 可以更加容易地实现数据绑定、样式和模板等功能。

WPF 是一种用于创建 Windows 桌面应用程序的技术，它是 .NET Framework 的一部分。WPF 提供了基于 XAML 的声明性方式来创建用户界面，并且提供了一些强大的功能，例如数据绑定、模板、样式、动画、3D 渲染等。

2.2 代码实现

我将项目分为了三个子模块进行设计，包括：堆排序，快速排序和冒泡排序。从而使程序能够适用于不同需求的用户，实现不同的功能。选择这三个排序的主要原因是想测试不同时间复杂度类型的排序算法的实际性能。

因此，在代码框架部分，我设计了四个窗口，主界面（MainWindow），和另外三个分界面（HeapSort，QuickSort，BubbleSort）。同时，项目文件中也包括了一个图标文件夹（icon）和程序运行所依赖的相关项（Reference），及包文件（packages）。



2.2.1 MainWindow

在主界面部分，主要包括三个模块，分别是标题栏，消息栏和内容栏。同时设置了三个入口，可以进入不同的功能分区。当鼠标指针移动到某个按钮上方时，下方的消息栏会提示相关信息，指导用户进行选择。

从艺术设计上讲，顶部的标题栏背景使用了渐变的紫色，体现出沉稳大气、雍容华贵的特点。（其中淡紫色、黛紫色、深紫色的比例也都各是黄金分割比1：0.618）。三个功能按钮的图标采用统一的拟物风格，具有卡通特色，且图标与各自的特点相契合。程序主体采用深灰色的背景，格外严谨而庄重。

窗口右下角部分为作者标识，绿色与黄色交织，常春藤缠绕在其上。三叶草象征着自由与希望。也寓示着绿色软件与自由软件的发展将生机勃勃。

2.2.2 Sort

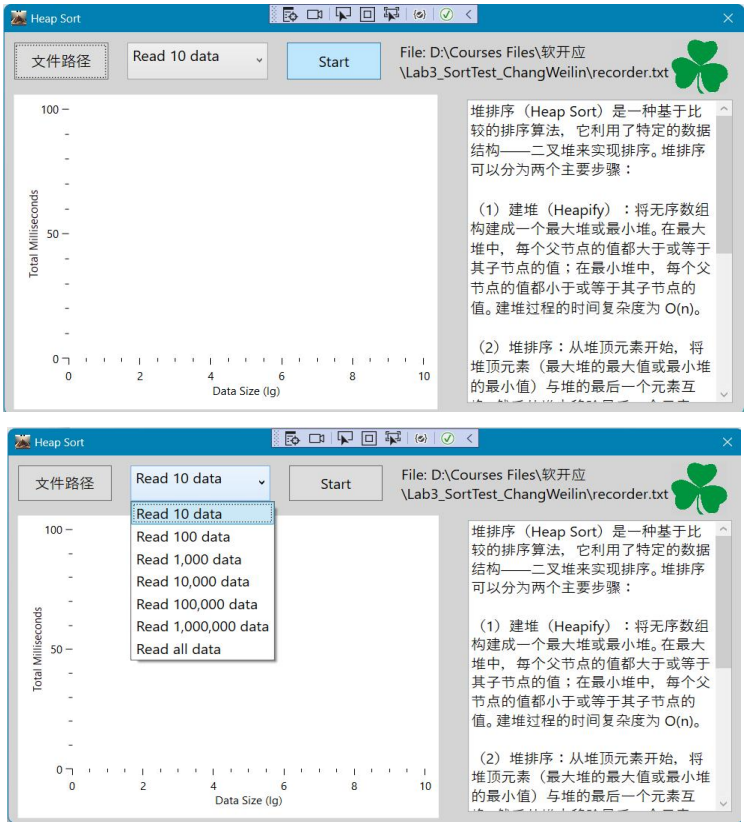
在排序界面，我设置如下功能：

（1）读取用户选定的文件，并在读取之后将对应文件的绝对路径显示在右上角的信息提示栏。

（2）允许用户选择进行排序的数据规模，这样做的目的是可以便于控制变量进行实验。允许用户选择 10、100、1000、10000、100000、1000000 或全部数据。当选择全部数据时，即使读入数据规模比 100 万小，也能够显示准确的排序数据量。

（3）点击 start 按钮后即开始排序，排序结束后会将对应的点显示在坐标系中，当添加两个以上的点时，会依据横坐标的大小将多个点连成折线。

（4）由于横坐标数据规模和纵坐标排序时间相差数量级较大，因此将横坐标取为 10 的对数坐标，这样显示的横坐标就是数据规模的数量级（10 的 n 次方）而不是实际的数据量。



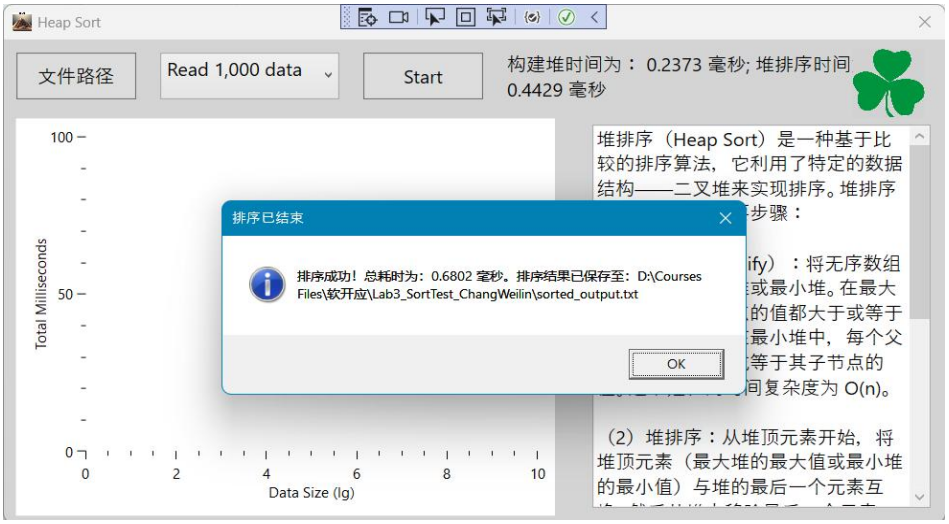
（5）同时在右侧设置了一个静态文本框用于显示当前排序算法的一些文字描述信息和时间复杂度。以便于用户对排序算法更加了解。

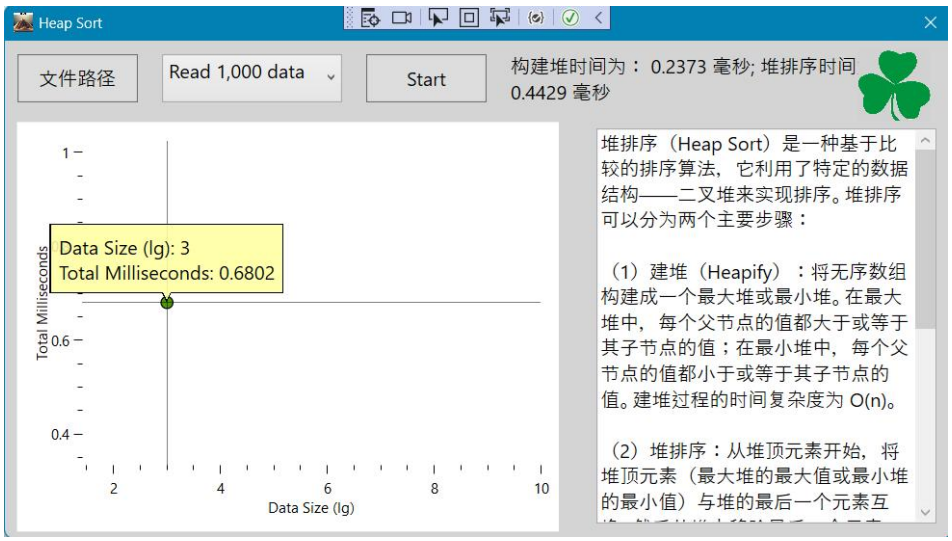
在排序完成后，会弹出提示窗，显示排序所耗时间，并将排好序的文本按照原格式输出到同目录下的文件中，并会显示排序文件的绝对路径。

此时，对应的点也会被添加到坐标系中。例如：此时显示数据规模为 3，表示 10 的 3 次方，也就是 1000，排序时间为 0.68 毫秒，非常快，不到 1 秒钟的 1/1000。（这个时间仅包括排序算法的执行时间，而不包括文件读写的时间）

对于堆排序而言，还会专门显示构建堆和进行堆排序的时间。最终总排序时间是这两者的加和。对于其他排序，则只会显示一个总的排序时间。

以上是堆排序界面的展示，其他排序窗口与之类似，故不做展示，在测试和性能分析部分会具体对三种算法再做分析。

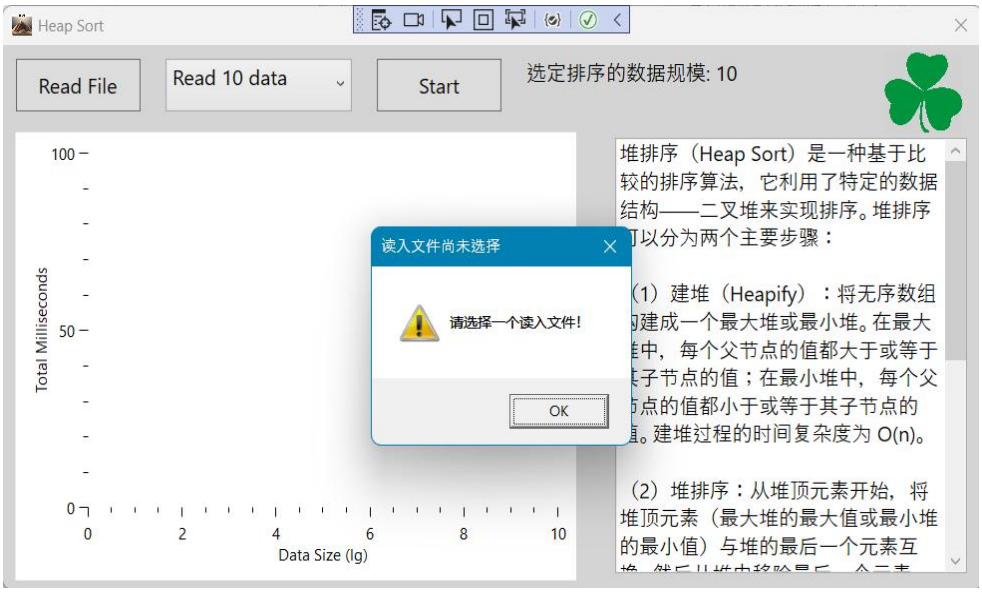




三、 实验结果

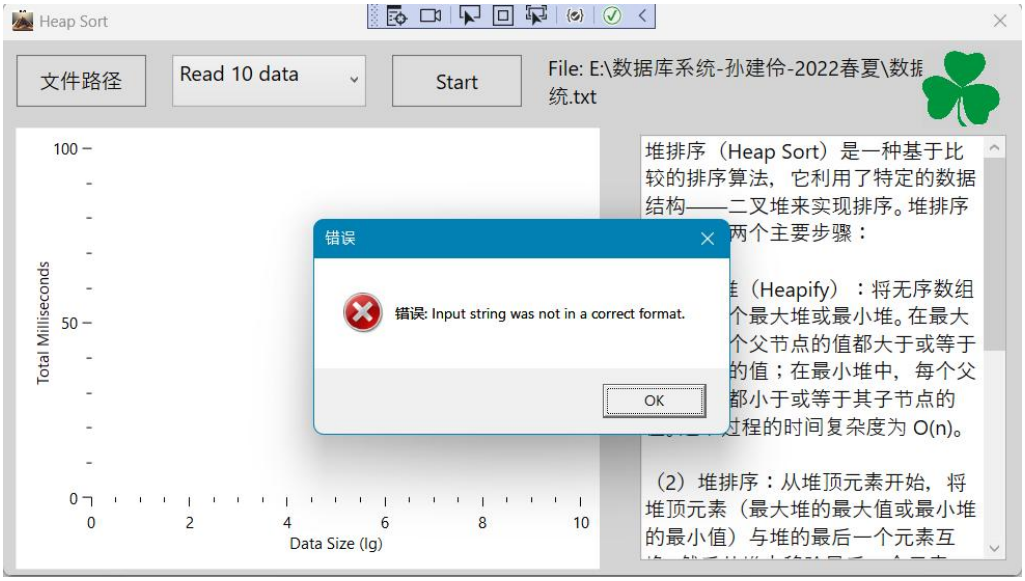
3.1 异常测试

用户在设置排序算法时，必须指定读入的待排序文件以及排序的数据规模。（默认为 10），否则会弹出警告窗口，提示用户没有选择读入文件。



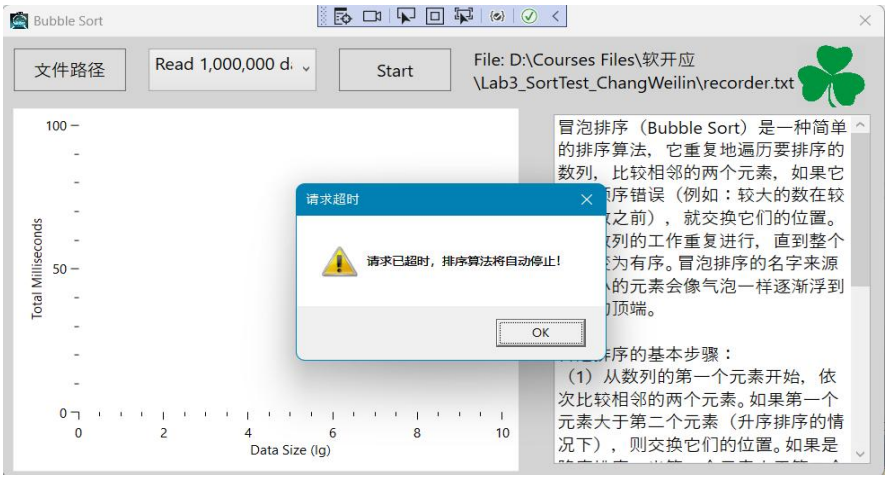
如果用户读入的待排序文件中的数据格式不正确，则会弹出一个 Error，告知用户所选择的文件中数据格式错误。

本程序要求输入的文件必须为文本文件，同时数据之间使用一个空格间隔。



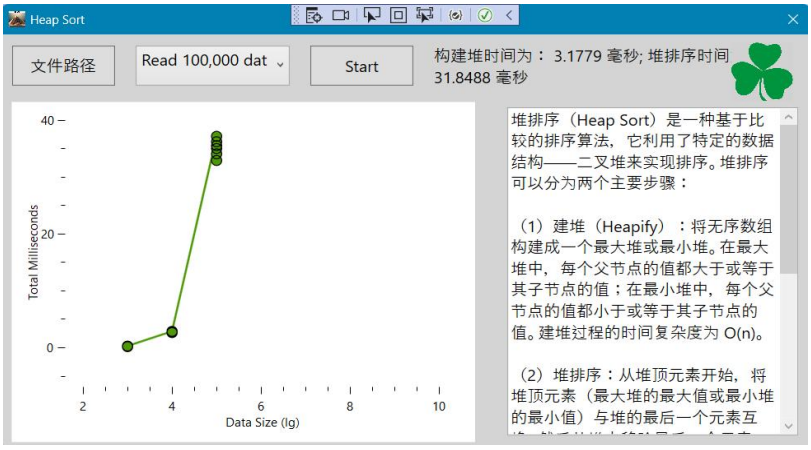
对于某些算法（如冒泡排序），当排序时间过长（超过 1 分钟）仍然没有完成排序时，则会立刻终止排序，避免出现死锁导致程序崩溃。

例如，使用冒泡排序对 100 万数据规模的数据进行排序，理论上按照 O(n^2)的时间复杂度，需要比 O(nlogn)的时间复杂度花费近 500 倍的时间，因此我设计当排序时间超过 60s（也就是 60000ms 仍然没有完成排序后，立刻终止排序，返回一个提示）。



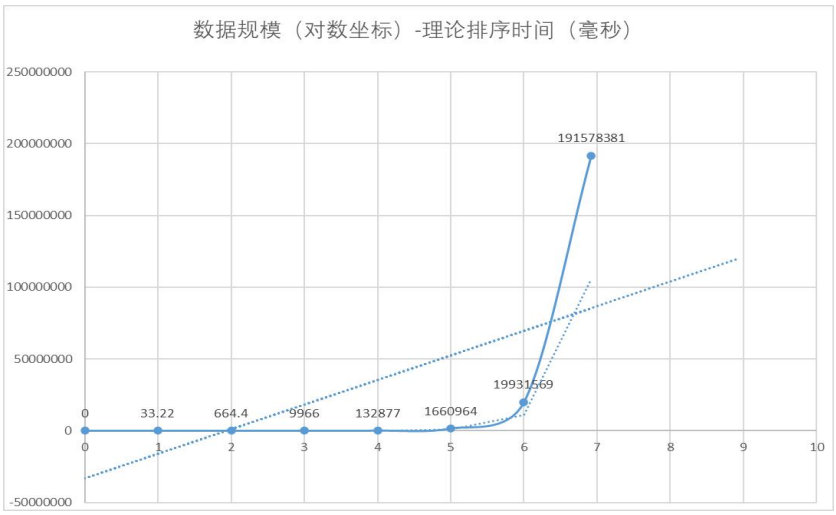
3.2 算法测试

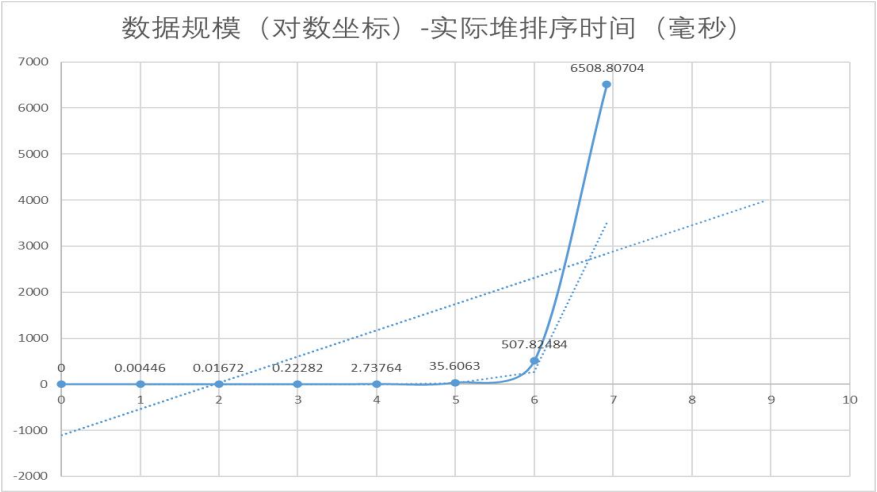
在算法测试部分，我主要进行了理论性能分析和实际性能测试。每个算法的每个样本点取五次测试成绩的平均值。具体测试数据详见 Excel 表格。



3.2.1 HeapSort：

（详见 Excel 表格中的数据）





A	B	C	D	E	F	G	H	I	J
堆排序 O(nlog2(n))									
数据规模	对数数据规模	理论排序时间	实际排序时间1	实际排序时间2	实际排序时间3	实际排序时间4	实际排序时间5	平均实际排序时间	
1	0	0	0	0	0	0	0	0	
10	1	33.22	0.0044	0.0047	0.005	0.0038	0.0044	0.00446	
100	2	664.4	0.0159	0.0158	0.0192	0.0169	0.0158	0.01672	
1,000	3	9966	0.3063	0.1986	0.2017	0.2024	0.2051	0.22282	
10,000	4	132877	2.8885	2.7084	2.7368	2.6723	2.6822	2.73764	
100,000	5	1660964	37.1681	34.0915	35.529	36.2162	35.0267	35.6063	
1,000,000	6	19931569	503.8996	500.3796	500.2788	513.4838	521.0824	507.82484	
8,332,973	6.9208	191578381	6753.0623	6113.3377	6509.0804	6369.97	6798.5848	6508.80704	
数据规模	对数数据规模	理论排序时间	建堆时间1	建堆时间2	建堆时间3	建堆时间4	建堆时间5	平均建堆时间	占总排序时间的比例
1	0	0	0	0	0	0	0	0	-
10	1	33.22	0.0037	0.0039	0.0042	0.0031	0.0037	0.00372	83.40%
100	2	664.4	0.0055	0.0053	0.0074	0.0061	0.0053	0.00592	35.40%
1,000	3	9966	0.1348	0.0277	0.0286	0.0284	0.0289	0.04968	22.30%
10,000	4	132877	0.257	0.255	0.2595	0.2555	0.2769	0.26078	9.53%
100,000	5	1660964	2.5356	2.6117	2.6775	2.7836	3.1779	2.75726	7.74%
1,000,000	6	19931569	31.1206	29.077	30.3411	28.2663	27.7588	29.31376	5.77%
8,332,973	6.9208	191578381	250.7548	249.114	286.2464	254.0877	277.501	263.54078	4.05%
数据规模	对数数据规模	理论排序时间	堆排序时间1	堆排序时间2	堆排序时间3	堆排序时间4	堆排序时间5	去掉建堆时间后的平均堆排序时间	占总排序时间的比例
1	0	0	0	0	0	0	0	0	-
10	1	33.22	0.0007	0.0008	0.0008	0.0007	0.0007	0.00074	16.60%
100	2	664.4	0.0104	0.0105	0.0118	0.0108	0.0105	0.0108	64.60%
1,000	3	9966	0.1715	0.1709	0.1731	0.174	0.1762	0.17314	77.70%
10,000	4	132877	2.6315	2.4534	2.4773	2.4168	2.4053	2.47686	90.47%
100,000	5	1660964	34.6325	31.4798	32.8515	33.4326	31.8488	32.84904	92.26%
1,000,000	6	19931569	472.779	471.3026	469.9377	485.2175	493.3236	478.51208	94.23%
8,332,973	6.9208	191578381	6502.3075	5864.2237	6222.834	6115.8823	6521.0838	6245.26626	95.95%

根据以上测试数据，发现堆排序算法的一些特点：

（1）随着数据规模的增加，平均实际排序时间和理论排序时间都呈现出增长的趋势，这与堆排序的时间复杂度 O(nlog2(n))相符。

（2）建堆时间在总排序时间中占比随着数据规模的增加而减少，这表明建堆操作在整个堆排序过程中的影响逐渐减小。例如，当数据规模为 10 时，建堆时间占总排序时间的比例为 83.40%，而当数据规模达到最大时（800 万），这一比例降至 4.05%。

（3）在去掉建堆时间后，剩余堆排序时间占总排序时间的比例随着数据规模的增加而增加。例如，当数据规模为 10 时，去掉建堆时间后的平均堆排序时间占总排序时间的比例为 16.60%，而当数据规模达到最大，这一比例升至 95.95%。

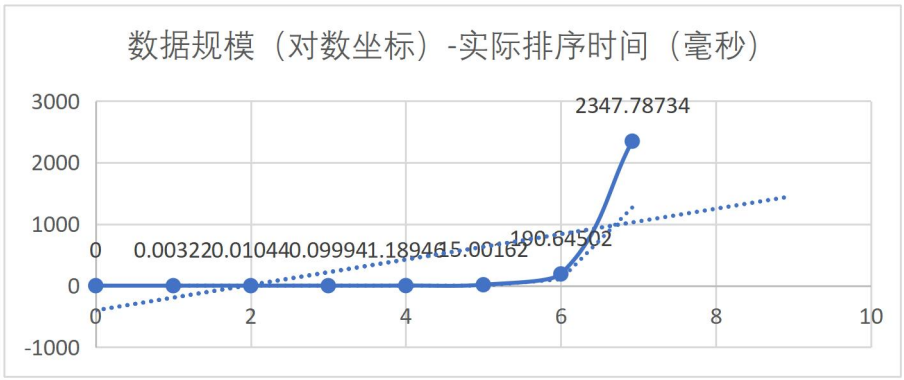
（4）从给出的数据中，可以观察到实际排序时间与理论排序时间之间的差距。理论排序时间是基于堆排序的时间复杂度 O(nlog2(n))计算得出的，而实际排序时间是从实际运行结果中得出的。在大部分情况下，实际排序时间要低于理论排序时间。这是因为实际运行环境中存在一定的优化，使得堆排序的性能得到了提升。

（5）对于较小规模的数据来讲，堆排序的建堆时间会对总体排序时间造成一定的影响，这在之后的与快速排序的对比中也得以体现。但是堆排序对于大规模数据的处理较好，尤其是处理动态变化的排序问题时，能够体现“堆”数据结构的优势。

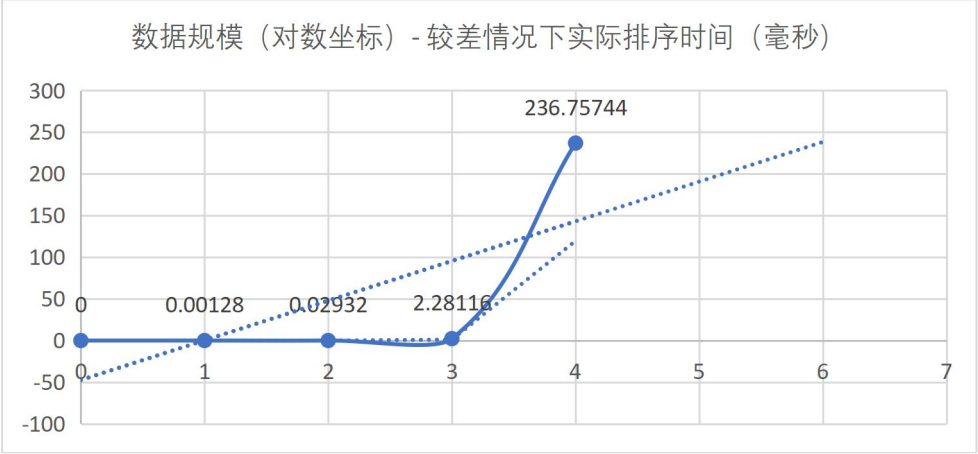
3.2.2 QuickSort：

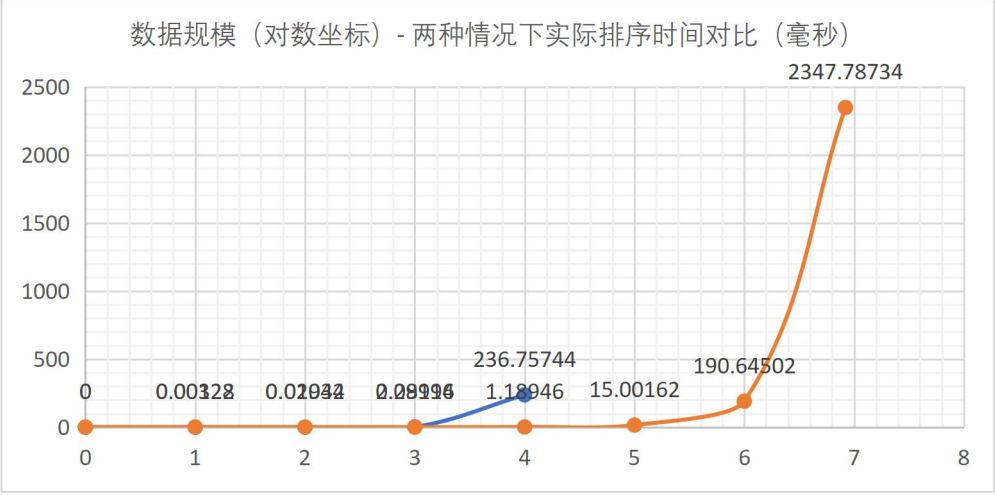
(详见 Excel 表格中的数据）

快速排序的时间复杂度介于 O(nlogn)和 O(n^2)之间。不过对于现实中大多数情况来讲，很少会遇到快速排序退化后的情况。本次实验，对快速排序在随即情况和最坏情况下的性能都进行了测试，对比发现性能差异较大。不过在实际使用过程中，仍然可以对快速排序进行优化，以防止出现退化的情况。例如：TimSort 就是目前相对而言很好的一种广泛适用的排序算法。


数据规模（对数坐标）-实际排序时间（毫秒）
3000
2000
1000
0
-1000
0
2
4
6
8
10
2347.78734
190.64502
15.00162
1.18946
0.099941
0.010440
0.003220
0


数据规模（对数坐标）-理论排序时间（毫秒）
250000000
200000000
150000000
100000000
50000000
0
-50000000
0
2
4
6
8
10
191578381
19931569
1328771660964
9966
664.4
33.22
0


数据规模（对数坐标）- 较差情况下实际排序时间（毫秒）
300
250
200
150
100
50
0
-50
-100
0
1
2
3
4
5
6
7
236.75744
2.28116
0.02932
0.00128
0


数据规模（对数坐标）- 两种情况下实际排序时间对比（毫秒）
2500
2000
1500
1000
500
0
0
1
2
3
4
5
6
7
8
2347.78734
236.75744
190.64502
15.00162
1.18946
0

快速排序 O(nlog2(n)~n^2)								
数据规模	对数数据规模	理论排序时间	实际排序时间1	实际排序时间2	实际排序时间3	实际排序时间4	实际排序时间5	平均实际排序时间
1	0	0	0	0	0	0	0	0
10	1	33.22	0.0037	0.0035	0.0031	0.0021	0.0037	0.00322
100	2	664.4	0.0097	0.0115	0.0105	0.0107	0.0098	0.01044
1,000	3	9966	0.0968	0.0985	0.0973	0.1075	0.0976	0.09994
10,000	4	132877	1.2019	1.1822	1.1796	1.1984	1.1852	1.18946
100,000	5	1660964	14.609	16.0738	14.6484	14.7304	14.9465	15.00162
1,000,000	6	19931569	186.1084	183.4682	183.5533	199.8839	200.2113	190.64502
8,332,973	6.9208	191578381	1831.5331	2461.9808	2831.128	2345.3244	2268.9704	2347.78734

数据规模	对数数据规模	理论排序时间	较差的快速排序1	较差的快速排序2	较差的快速排序3	较差的快速排序4	较差的快速排序5	较差情况下的平均实际排序时间
1	0	1	0	0	0	0	0	0
10	1	100	0.0019	0.0011	0.0012	0.0011	0.0011	0.00128
100	2	10000	0.0254	0.0393	0.0257	0.0282	0.028	0.02932
1,000	3	1000000	2.147	2.3354	2.3014	2.1003	2.5217	2.28116
10,000	4	100000000	219.3295	232.9398	238.5859	253.7645	239.1675	236.75744
100,000	5	10000000000	Timeouts	Timeouts	Timeouts	Timeouts	Timeouts	Timeouts
1,000,000	6	1E+12	Timeouts	Timeouts	Timeouts	Timeouts	Timeouts	Timeouts
8,332,973	6.9208	69,438,439,018,729	Timeouts	Timeouts	Timeouts	Timeouts	Timeouts	Timeouts

根据以上测试数据，发现快速排序算法的一些特点：

（1）快速排序的平均时间复杂度为 O(nlogn)。在最佳情况下，每次都能选择到中位数作为基准值，此时时间复杂度为 O(nlogn)。在最差情况下，每次都选择到最小值或最大值作为基准值，此时时间复杂度为 O(n^2)。

（2）然而，通过一些优化方法（例如随机选择基准值或使用三数取中法）可以降低最差情况出现的概率，使得快速排序在实际应用中具有更好的性能，因此对绝大部分排序需求，使用快速排序往往是最好的解决方法。

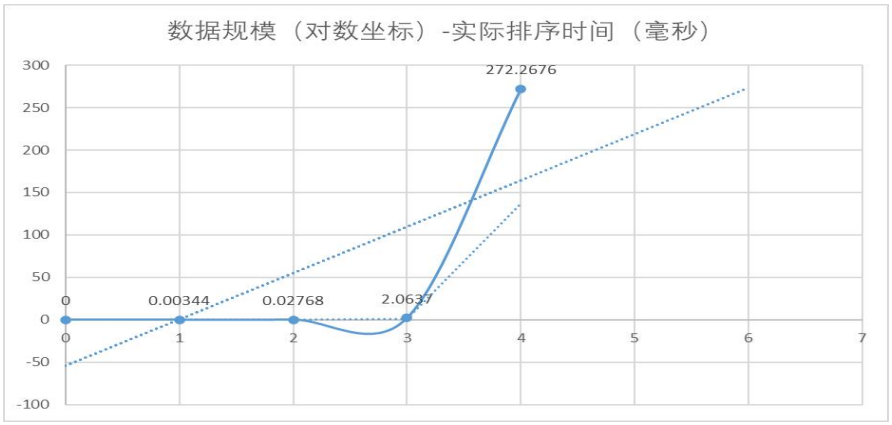
（3）对于快速排序，理论排序时间的范围是 O(nlog2(n))到 O(n^2)，这意味着快速排序的性能可能受到数据分布和初始状态的影响。

（4）在正常情况下（即数据分布较好或随机化的情况下），快速排序的实际排序时间随着数据规模的增加而增加，与堆排序类似。在给出的数据中，实际排序时间与理论排序时间 O(nlog2(n))较为接近。这说明快速排序在正常情况下表现出较好的性能。

（5）在较差情况下（如已排序或接近排序的数据集），快速排序的性能将受到显著影响。从给出的数据中，可以看到较差情况下的平均实际排序时间远高于正常情况下的实际排序时间。特别是当数据规模较大时，较差情况下的快速排序性能下降至超时（Timeouts）。这说明在较差情况下，快速排序的性能可能会退化至 O(n^2)的复杂度。

3.2.3 BubbleSort：

（详见 Excel 表格中的数据）



水平 (值) 轴 次要网格线

冒泡排序 O(n^2)

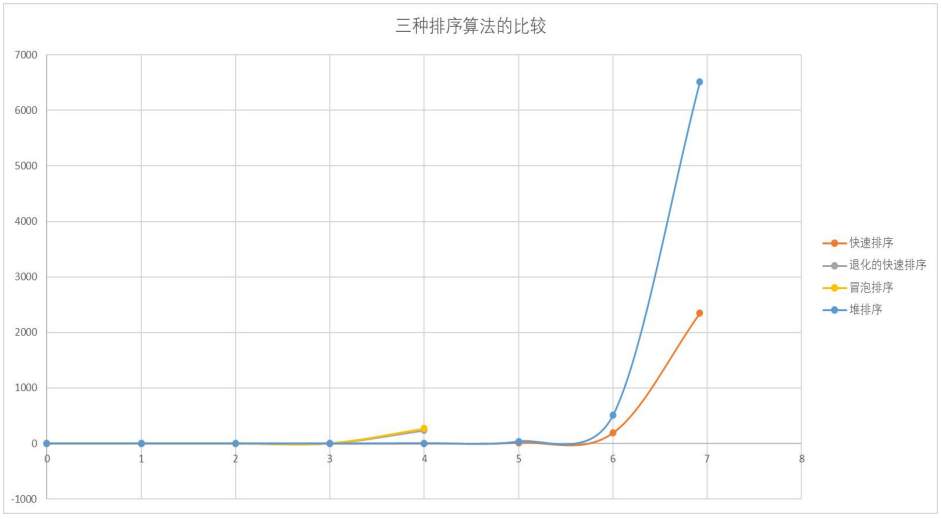
数据规模	对数数据规模	理论排序时间	实际排序时间1	实际排序时间2	实际排序时间3	实际排序时间4	实际排序时间5	平均实际排序时间
1	0	1	0	0	0	0	0	0
10	1	100	0.0038	0.0036	0.0032	0.0034	0.0032	0.00344
100	2	10000	0.0262	0.0303	0.0292	0.0271	0.0256	0.02768
1,000	3	1000000	2.0148	2.0246	2.0096	2.0757	2.1938	2.0637
10,000	4	100000000	281.1179	269.6697	268.789	269.9215	271.8399	272.2676
100,000	5	10000000000	Timeouts	Timeouts	Timeouts	Timeouts	Timeouts	Timeouts
1,000,000	6	1E+12	Timeouts	Timeouts	Timeouts	Timeouts	Timeouts	Timeouts
8,332,973	6.9208	69,438,439,018,729	Timeouts	Timeouts	Timeouts	Timeouts	Timeouts	Timeouts

根据以上测试数据，发现冒泡排序算法的一些特点：

（1）在数据规模较小的情况下，实际排序时间远低于理论排序时间。随着数据规模的增加，实际排序时间趋近于理论排序时间。

（2）当数据规模进一步增加至 100,000 或更高时，冒泡排序的执行时间变得非常长，导致超时。证明冒泡排序在大规模数据上的性能较差，对于较大规模的排序，需要寻找其他更高效的排序算法。

3.2.4 总结：



数据规模	堆排序平均排序时间	快速排序平均排序时间	退化的快速排序平均排序时间	冒泡排序平均排序时间
1	0	0	0	0
10	0.00446	0.00322	0.00128	0.00344
100	0.01672	0.01044	0.02932	0.02768
1,000	0.22282	0.09994	2.28116	2.0637
10,000	2.73764	1.18946	236.75744	272.2676
100,000	35.6063	15.00162	Timeouts	Timeouts
1,000,000	507.82484	190.64502	Timeouts	Timeouts
8,332,973	6508.80704	2347.78734	Timeouts	Timeouts

对于大多数情况，快速排序和堆排序更适合处理大量数据的排序问题。特别是对于快速排序，如果可以通过随机化选择基准元素或者使用其他优化措施避免最坏情况的发生，那么它的性能会非常出色。而冒泡排序适用于小规模数据集或者需要稳定排序算法的场景。

四、实验评价及总结

本次实验我使用.NET 框架和 C# 语言开发一个排序算法的测试程序。实验使用 XAML 和 C# 语言完成,实现对堆排序、快速排序、冒泡排序算法性能的测试。主要测试算法在最好情况、最坏情况和平均情况下的时间复杂度和稳定性。开发过程中，我也遇到许多困难,但通过不断尝试学习,最终完成实验要求。

首先，测试数据量较大时,算法的运行时间较长,难以多次运行获得准确结果。需要对测试数据量进行控制,确保算法在合理时间内完成。

其次，测试环境的硬件配置会影响算法的运行效率,无法保证每个人的测试环境完全相同,实验结果可能受环境影响。需要在同一台机器上进行多次测试取平均值。同时算法实现中难免产生错误,需要仔细检查代码,确保实现正确无误。

本次实验我认为优点在于：提供了图形化界面，实验设计简单清晰,容易理解,且实验结果直观。可以清楚观察不同算法在各种情况下的性能表现。实验有一定的通用性,不仅适用于本次课题的排序算法,也可扩展到其他算法性能分析。

然而，本次实验因为开发时间有限，难免存在一些问题与不足之处。

第一，应该测试更大规模的数据,如几百万乃至上亿条记录,以使实验结果更加准确可信。但这需要更高性能的机器以支持大数据量测试。测试更多种排序算法,如堆排序、计数排序等,使实验更加全面。

第二，除时间复杂度外,也测试空间复杂度,并根据实际应用场景选择最优算法。同时，除单核环境下的测试外,也可以测试多核 CPU 下的并行算法,观察并行环境下的性能提升。

实验过程充满挑战,但最终能够完成所有功能,对我来说收获很大。虽然程序还不够完善,但是通过分析问题、查阅资料并不断实践的过程,让我对程序开发有了更深的体会。通过本次实验,我的编程能力和逻辑思维能力都得到了提高。

未来还需要对在现代项目中如何搭建框架、如何组织文件、如何继承多态、如何命名变量、如何使用 CMAKE、如何单元测试有进一步的认识和体会。

4/10/2023

（完）