苏 州 大 学 实 验 报 告

院、系	计算机学院	年级专业	19 计算机类	姓名	张昊	学号	1927405160
课程名称	数据结构课程实践					成绩	
指导教师	孔芳	同组实验者	无		实验日期	2020 年 12 月 14 日	

实 验 名 称 排序算法的实现及性能测试及比较

一、问题描述及要求

排序算法的实现及性能测试及比较

在书中，各种内部排序算法的时间复杂度分析结果只给出了算法执行时间的阶，或大概执行时间。试通过具体数据比较各种算法的关键字比较次数和记录移动次数，以取得直观感受。

要求:

(1)编写程序创建一些整数文件用于排序。创建的这些文件可以根据需要生成不同的长度，如长度分别为 20，200 和 2000，以正序、逆序、随机顺序的方式创建这些文件，通过把所有这些测试数据保存在文件中(而不是每次在测试程序时用随机数生成)，可以使用同样的数据去测试不同的方法，因此会更易于比较这些方法的性能。

(2)数据表采用顺序存储结构，实现插入排序，选择排序，希尔排序，归并排序，快速排序，堆排序，冒泡排序，并对这些算法的实现效率进行比较和分析。

(3)排序的指标包含：运行时间，比较次数，移动次数。

二、概要设计

1. 问题简析

本次实验内容是排序算法的实现及性能测试及比较。实验需要生成一批大小不同的数据文件，使用这些文件生成不同大小的顺序线性表，分别通过对同一组数据元素的不同顺序调用同一排序算法，对不同组数据元素的同一顺序调用各个排序算法，分析算法的实际执行时间、统计执行过程中的比较次数和元素的移动次数，以此来测试和比较各种排序算法的性能。

2. 系统功能列表

程序要实现如下功能：

1. 创建一些整数文件用于排序；
2. 实现插入排序，选择排序，希尔排序，归并排序，快速排序，堆排序，冒泡排序算法；
3. 在上面实现的排序算法中添加计算执行时间、统计比较次数和移动次数的功能；
4. 对这些排序算法的实现效率进行比较和分析；

3. 程序结构设计思路

程序将数据组织为一个顺序线性表类，并且为类添加不同的排序函数。首先由生成数据集的程序创建整数文件。在排序顺序表类中，通过给定的正序、逆序、随机顺序以及大小读取相应的数据文件，并调用依次不同的排序算法。各个排序算法中计算执行时间、统计比较次数和移动次数，并将结果以三元组是形式返回。主函数按找顺序类型和大小依次读取每个数据文件，并依次调用排序算法多次，几次排序所用时间取平均值，并将结果写入文件。

在 C++中，系统提供的计时函数返回结果的单位并不适合这一程序，故程序包装了系统在头文件<ctime>中提供的有关计时的函数，作为 Timer 类供计时使用。

三、详细设计

（一）整数文件的创建

注：这一部分在整个程序中只需一次且多次写入文件，故使用 Python 脚本实现。

程序的目的是创建一些整数文件用于排序。根据需要，程序按照 30，300，3000，30000 的数据规模，以及正序、逆序、随机顺序来创建多组不同的整数文件。

程序每次生成数据时，以提供的数据规模为基础，乘以一个质数（这里选用 11）作为数据的范围，在此范围内产生随机数生成一个大小为数据规模的整数列表。并将其排序，分别存储随机顺序、正序、逆序到不同的文本文件中。

```
data_set = [random.randint(0, int(11 * size)) for _ in range(size)]
```

这样，通过把所有这些测试数据保存在文件中，可以使用同样的数据去测试不同的方法。

（二）排序算法的实现

1 插入排序

直接插入排序的基本思想是依次将待排序序列中的每一个记录插入到已排好序的序列中，直到全部记录都排好序。首先是构造初始有序序列。将第 1 个记录看成是初始有序序列，然后从第 2 个记录起依次插入到有序序列中，直至将第 n 个记录插入。

一般来说，在有序区 `data[0..i-1]`中插入记录 `data[i]`时，首先要找到正确插入位置。这里采用顺序查找的方法。设下标 `j` 从 `i-1` 往前查找插入位置，同时后移记录，为了避免覆盖待插入记录，将 `data[i]`用临时变量 `temp` 暂存，循环条件为 `temp<data[j]`。退出循环就说明找到了插入位置，`j+1` 为正确的插入位置，将 `temp` 暂存的记录存储到 `data[i+1]`中即可。

2 简单选择排序

简单选择排序的思想是第一次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，然后再从剩余的未排序元素中寻找到最小（大）元素，然后放到已排序的序列的末尾。以此类推，直到全部待排序的数据元素的个数为零。

n 个记录的简单选择排序可经过 n-1 趟简单选择排序得到有序结果：初始时无序区为 data[0..n-1]，有序区为空。在无序区 data[0..n-1]中选出关键字最小的记录 data[k]，将它与无序区的第 1 个记录 data[0]交换，使 data[0]和 data[1..n]分别变为记录个数增加 1 个的新有序区和记录个数减少 1 个的新无序区。第 i 趟排序开始时，当前有序区和无序区分别为 data[0..i-1]和 data[i..n]。该趟排序从当前无序区中选出关键字最小的记录 data[k]，将它与无序区的第 1 个记录 data[i]交换。

3 希尔排序

希尔排序的基本思想是将待排序序列分割成若干个子序列，在子序列内分别进行直接插入排序，待序列基本有序时，对全体记录进行直接插入排序。

这里以 d 为增量，在每个子序列内进行直接插入排序。在每个子序列中，待插入录和同子序列中的前一个记录比较，在插入 data[i]时，自 data[i-d]起以幅度 d 往前跳跃式查找待插入位置，在查找过程中，记录后移也是跳跃 d 个位置，当搜索位置 j<0 或者 temp≥data[j]表示插入位置已找到，退出循环，j+d 为正确的插入位置。在整个序列中，记录 data[0..d-1]分别是 d 个子序列的第一个记录，所以从记录 data[d]开始进行插入。

4 冒泡排序

冒泡排序的思想是重复地走访过要排序的元素序列，依次比较两个相邻的元素，如果顺序误就把他们交换过来。走访元素的工作是重复地进行直到没有相邻元素需要交换，也就是说该元素序列已经排序完成。其实现细节不在赘述。

5 快速排序

快速排序的基本思想是选一个枢轴量，将待排序记录划分成两部分，左侧记录均小于或等于枢轴量，右侧记录均大于或等于枢轴量，然后分别对这两部分重复上述过程，直到整个序列有序。

这里取第一个记录作为枢轴量。以枢轴量为基准将无序序列划分为两部分，较大的记录移到后面，较小记录移到前面。设待划分的序列存储在 data[first..last]中，j 从后向前扫描，直到 data[j]<data[i]，交换 data[j]和 data[i]，i++；i 从前向后扫描，直到 data[j]<data[i]，交换 data[j]和 data[i]，j--；重复上述过程，直到 i 等于 j。递归执行快速排序，一次划分后分别对左侧、右侧子序列进行快速排序。

6 归并排序

归并排序的基本思想是将待排序序列划分为两个长度相等的子序列，分别对这两个子序列进行排序，得到两个有序子序列，再将这两个有序子序列合并成一个有序序列。

由于两个有序子序列的合并过程会破坏原有序列，合并不能就地进行。合并两个序列的方法是竞争插入新的序列中，待其中一个序列达到末尾时，将另一个序列剩余

元素依次插入到新的序列中。最后将系的序列覆盖掉原来的两个序列即可。归并排序采用递归的方式进行，当待排序区间中只有一个记录时结束，否则将待排序序列均分成两个等长的序列，分别在前后两个子序列中归并排序，并将两个已排序的子序列合并。

7 堆排序

堆排序的基本思想是首先将待排序序列构造成一个大根堆，即选出了堆中所有记录的最大者，将它从堆中移走，并将剩余的记录再调整成堆，这样又找出了次小的记录，以此类推，直到堆中只有一个记录。

初始建堆的过程就是反复调用堆调整的过程。序列对应完全二叉树的顺序存储，所有叶子结点都已经是堆，只需从最后一个分支结点到根结点，执行堆调整。初始建堆后，将待排序序列分成无序区和有序区两部分：无序区对应一个大根堆且包括全部待排序记录，有序区为空。将堆顶与堆中最后一个记录交换，则堆中减少了 1 个记录，有序区增加了 1 个记录.第 `i` 趟`(1≤i≤length-1)`堆排序对应的堆中最后一个记录是 `data[length-1]`,将 `data[i]`与 `data[length-1]`交换。第 `i` 趟排序后，无序区有 `length-i` 个记录，在无序区对应的完全二叉树中只需调整根结点即可重新建堆。

（三）排序指标的获取

为了对不同的排序算法有直观的感受，测试各算法之前预先生成了一批数据，数据规模覆盖 30~30000，并对每组数据采取正序、逆序、随机顺序三种方式存储。通过对同一数据调用不同的排序算法、同一排序算法使用不同大小和数据顺序测试，将结果记录在文本文件中。

本实验中对排序结果的评估采用了如下三个指标：算法的实际执行时间、算法执行过程中元素的比较次数和元素的移动次数。为了更好地组织这三个指标，将其封装为结构体 `Information`，每个算法对类中的数据元素执行相应的排序算法，执行结束后会返回一个结构体对象。由于计算机在不同时刻的运行速度可能存在误差，主函数中对于同一算法、同一数据集重复调用 5 次，对得到的运行时间取平均值，作为最终的结果，保存到文件中。对于需要调用多个辅助函数以及递归调用的算法，采用了传递引用参数的方法来避免结构体对象的多次复制从而导致影响时间测定的准确性。

实验中时间的测量利用了系统头文件<ctime>中提供的有关计时的函数，并对齐进行了简单的封装（如下所示），以避免多次复制粘贴相同的计算时间的代码。

```
1.  class Timer {
2.   public:
3.    Timer() {
4.      time_ = std::clock();
5.    }
6.    double runtime() const {
7.      return (double) (std::clock() - time_) / CLOCKS_PER_SEC;
```

```
8.    }
9.   private:
10.   std::clock_t time_;
11. };
```

在算法中，为了计算执行过程中元素的比较次数和元素的移动次数，在实现各排序算法的同时添加计数语句（如下），同时利用逗号表达式使这一过程更为简洁。

记录比较次数	`information.compare++, data_[i] <= data_[j]`
记录交换次数	`std::swap(data_[i], data_[j]), information.move++;`

四、实验结果测试与分析

经过测试，各个排序算法在不同数据规模，分别在正序、逆序、随机顺序情况下的指标如第 6 页表格 1 所示（测试用的数据集保存在附件 src/out/data 文件夹，结果保存在附件 src/out 文件夹）。

为了有更直观的感知，估计了理论上各时间复杂度的执行时间（见表格 1 的附表，以 30000 规模下逆序和随机的冒泡排序结果记，仅数量级有效），并利用 Excel 的条件格式为不同数量级的数据涂上不同的颜色。从表格中可以看到，当数据规模较小时（30），各排序算法的绝对运行时间相差不大，最多相差一个数量级，且正序、逆序和随机顺序对排序所消耗的时间没有太大影响。

当数据规模增大后，插入排序和冒泡排序在正序的情况下为线性阶，为最好情况；而到了逆序和随机顺序则为平方阶，为最差情况，这与最差情况下两种算法都在频繁移动元素有关。选择排序由于不能识别有序的数据，做了大量的比较，使其排序时间性能为测试的所有算法中在所有情况下均为最差的，为平方阶。快速排序在随机顺序下表现最好，时间性能为$O(nlog_2n)$；但是其最坏情况为数据已经有序，时间复杂度退化为平方阶，这与大量的比较和递归调用有关，并且在这台计算机上用 300000 规模有序的数据测试快速排序时，程序由于递归深度多大，栈空间不足而导致了段错误，没能完整地测试 300000 规模下各算法的表现。归并排序、堆排序和希尔排序在测试中时间性能基本处于$O(nlog_2n)$，其中数据均没有达到希尔排序的最坏情况，故导致其表现和另外两种相近。

五、小结

本次实验对常用的排序算法进行了实现和系统性的测试，对这几种排序有了更加深入的了解和直观的感受。

六、附录

1. **源代码路径**：源代码位于附件中 src 目录下。　　2. **文件编码**：UTF-8；**行分隔符**：LF（\n）。
3. **实验环境**：Linux 操作系统，g++编译器，基于 cmake 构建；在 CLion 集成开发环境中调试运行通过。执行 build 文件并按屏幕提示运行程序，具体为（在 Linux/Unix shell 中）：

```
$ ./build  #pwd: src/
$ cd out
$ python3 ./generator.py  #生成数据（如果需要的话）
$ ./SortTester
```

4. **数据记录表**：

表格 1　排序算法在不同数据规模及顺序下的平均运行时间、元素比较次数以及移动次数记录表

数据集：附件 src/out/data 文件夹下各个文件

数据规模	顺序	插入排序			冒泡排序			选择排序			希尔排序			快速排序			堆排序			归并排序		
		运行时间	比较次数	移动次数	运行时间	比较次数	移动次数	运行时间	比较次数	移动次数	运行时间	比较次数	移动次数	运行时间	比较次数	移动次数	运行时间	比较次数	移动次数	运行时间	比较次数	移动次数
30	正序	0.000001	29	0	0.000002	29	0	0.000004	435	0	0.000002	94	0	0.000004	435	0	0.000006	211	134	0.000009	77	148
	逆序	0.000004	435	432	0.000010	435	432	0.000004	435	16	0.000003	136	67	0.000005	435	15	0.000005	188	107	0.000005	74	148
	随机	0.000003	214	188	0.000007	332	188	0.000005	435	23	0.000003	148	71	0.000004	121	42	0.000007	207	130	0.000008	112	148
300	正序	0.000003	299	0	0.000002	299	0	0.000168	44850	0	0.000015	2104	0	0.000177	44850	0	0.000077	4225	2409	0.000061	1308	2488
	逆序	0.000230	44849	44839	0.000923	44850	44839	0.000174	44850	155	0.000020	2812	1000	0.000151	44850	150	0.000075	3762	1996	0.000067	1189	2488
	随机	0.000101	21404	21108	0.000654	44835	21108	0.000186	44850	291	0.000032	3387	1430	0.000039	2704	870	0.000068	4025	2208	0.000071	2101	2488
3000	正序	0.000018	2999	0	0.000015	2999	0	0.013990	4498500	0	0.000183	30007	0	0.012943	4498500	0	0.000774	62846	34179	0.000541	18076	34904
	逆序	0.019282	4498498	4498374	0.079268	4498500	4498374	0.015514	4498500	1556	0.000227	42673	15651	0.013175	4498500	1499	0.001066	57544	29751	0.000492	16905	34904
	随机	0.009904	2228057	2225061	0.067545	4488812	2225061	0.016104	4498500	2992	0.000815	59496	31024	0.000602	38205	13623	0.001080	60242	32073	0.000734	30936	34904
30000	正序	0.000181	29999	0	0.000110	29999	0	1.503917	449985000	0	0.002577	390007	0	1.379536	449985000	0	0.011043	826532	440345	0.005623	227728	447232
	逆序	2.128544	449984959	449983662	7.638305	449985000	449983662	1.553159	449985000	15622	0.003346	566684	206660	1.353010	449985000	14999	0.009674	775684	399279	0.005403	220247	447232
	随机	0.947164	225358984	225328993	6.580225	449713238	225328993	1.392618	449985000	29984	0.007774	976141	601414	0.006342	511265	182056	0.012319	800639	419873	0.007926	408662	447232

附表：运行时间理论值估计（数量级有效）及图例

数据规模	30	300	3000	30000
$O(n)$	0.000000	0.000003	0.000027	0.000274
$O(nlog_2n)$	0.000001	0.000023	0.000316	0.004068
$O(n^2)$	0.000008	0.000821	0.082058	8.205828
时间估算				

执行一条语句的时间 (以 30000 规模下逆序和随机的冒泡排序的结果记) =运行时间/(比较次数+移动次数)
0.000000009