《数据结构》课程实践报告

院、系	计算机学院	年级专业	21 计算机科学与技术	姓名	赵鹏	学号	2127405037
实验布置日期	2022.10.25		提交日期	2022.11.16		成绩	

课程实践实验 7：排序算法的实现及性能测试及比较

一、问题描述及要求

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，堆排序等排序，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.问题分析

本次实验是排序算法的实现及性能测试及比较。首先需要生成一批不同规模、不同顺序的数据。根据这些数据生成不同的线性表。然后对生成的每一个线性表用不同的排序算法进行排序。记录下每种排序的比较次数、移动次数以及运行时间，以此测试不同排序算法在不同情况下的性能并对测试结果进行分析。

2.系统功能设计

为了实现测试各种排序算法的功能，系统需要实现以下的功能：

1. 生成不同规模、不同顺序的整数。
2. 实现冒泡排序、插入排序、选择排序、快速排序、归并排序、希尔排序、堆排序七种排序算法。
3. 在进行排序算法时记录下排序过程所需的运行时间、比较次数、移动次数。
4. 比较分析不同排序算法在不同情况下的运行效率。

3．程序结构设计

首先设计一个线性表类，为线性表添加不同的排序函数并实现。对于数据的生成部分，单独编写 C++程序，分别生成不同规模的随机整数，对于每种规模的数据，分别存储随机、顺序、逆序三种情况的文件。并为线性表添加从文件中读取数据的构造函数。然后读取每个数据文件，分别进行多种排序算法的测试，并在排序过程中记录运行时间、比较次数、移动次数即可。

三、详细设计

(一)数据文件的生成

此程序的目的是生成不同规模、不同顺序的一定整数供排序使用，故单独编写程序，运行一次即可。

在实现过程中，由于需要生成不同的数据规模(取 50、500、50000、50000)，故可以设定一个基底 base=50，对于每个规模生成顺序、逆序、随机三种情况的数据文件。生成完成后将 base 乘 10 后进行下一轮的生成即可。在实现过程中可以使用 C++11 中基于梅森旋转算法的 mt19937 快速产生高质量的伪随机数并存储到 vector 容器中，进行排序后即可得到顺序数据，对于逆序的情况，在排序后调用 algorithm 库中的 reverse 函数即可。随后将 vector 容器中的元素输出到文件中即可。

通过把测试数据保存在文件中，便可以使用同样的数据去测试不同的排序算法。

(二)排序算法的实现

1.冒泡排序

冒泡排序是一种较为简单的排序算法。它的思想是重复地走访过要排序的元素列，依次比较两个相邻的元素，如果顺序错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，也就是说该元素列已经排序完成。使用二重循环即可实现。时间复杂度为$O(n^2)$。

2.插入排序

插入排序的思想是把新元素不断的插入有序表中的特定位置，使插入后仍然有序，直至所有元素均插入完成为止。首先把第一个元素看为有序序列，然后从第二个元素开始依次插入到有序序列中，直到所有元素插入完成。

在有序序列 a[0,i-1]中插入元素 a[i]时，需要找到正确的插入位置，先用 temp 存储待插入元素 a[i],令下标 j=i-1 并不断向前查找，并不断将元素后移，直至 temp>=a[j],则说明找到了插入位置。j+1 即为正确的插入位置，将 temp 存储到 a[j+1]即可。

3.选择排序

选择排序的思想是第一次选择最小的元素放在序列开头作为初始序列，随后每次从未排序的序列中选择最小的元素加在已有序列的末尾。知道所有元素均被归到已排序序列中为止。

n 个元素的记录经过 n-1 次选择的过程即可完成排序。初始未排序序列为整个序列，已排序序列为空。在未排序序列中选择最小的放在已排序序列中作为第一个元素。第 i 次有序序列为 a[0,i-1],无序序列为 a[i,n]，通过依次遍历找到无序序列中的最小元素下标 k，交

换 data[k]与 data[i]即可。

4. 希尔排序

希尔排序的思想是将整个序列分割成若干个子序列,在每个子序列内分别进行插入排序,当整个元素基本有序时，对整个序列进行插入排序。

在实现过程中，可以 d 为增量划分出若干个子序列，在每个子序列内进行插入排序，即在插入 a[i]时依次与 a [i-d], a [i-2d]…进行比较，在插入时元素后移也同样时以 d 为间隔进行，比如 a[i]后移一次为 a[i+d]=a[i],在查找插入位置时当 temp>=a[j]或 j<0 表示找到插入位置，退出循环。j+d 即为正确插入位置。在整个序列中 a[0,d-1]为每个子序列的第一个部分，所以总 a[d]开始进行插入排序。

5. 快速排序

快速排序的思想是每次选取一个枢轴，把整个序列划分成两部分，左边的部分全都小于等于枢轴，右边的部分全都大于等于枢轴，即每次划分确定一个元素在排序后的位置。对左右两部分分别重复上述过程中，直至整个序列有序。

在实现时可以选取一个元素为枢轴量。以枢轴将序列划分成两个部分，较大的元素移到后面,较小的元素移到前面。设待划分序列为 a[left,right],令 i=left,j=right。j 从 right 向前扫描，直到 a[j]<a[i],交换 a[i]和 a[j]，并令 j--。再令 i 向后扫描，直至 a[i]>a[j],交换 a[i]和 a[j]，再令 i++。重复这个过程，直至 i 和 j 相等为止。再对左右部分递归执行这个过程中，即可完成排序。

6. 堆排序

堆排序的基本思想是将待排序序列建立为大根堆，即选出所有待排序元素中的最大者(堆顶元素)，从堆中移走，再将剩下元素调整为堆，不断此过程直至堆中只有一个元素。

初始化堆的过程中就是对已有无序序列调整成堆的过程中。整个序列对应一个完全二叉树的顺序存储，所有叶子结点已经是堆，所以只需要从最后一个分支到根开始调整即可。初始建立堆后将待排序元素分为有序和无序两部分,无序部分建立为大根堆并且初始包括所有元素，有序部分为空。每次将堆顶与堆中最后一个元素交换，减少堆中的一个元素，有序部分增加一个元素，第 i 次时堆中的最后一个元素为 a[length-1],将 a[i]与 a[length-1]交换。经过 i 次排序，有序部分有 i 个元素，无序部分有 length-i 个元素，重新调整无序部分为堆即可。

7. 归并排序

归并排序与快速排序同样有着划分的思想。将排序序列分为两个长度相等的子序列，分别对两个子序列进行排序，再将两个子序列合并成一个有序序列即可。

由于两个子序列的合并会破坏原有序列，所以需要开辟一个临时序列用来暂存合并结果。合并的方法是从两个有序序列中选取为合并的部分较小的元素插入临时序列中,当一个序列到达末尾时，将另一个序列的剩余元素直接插入临时序列中。最后将临时序列赋值给原序列即可。归并排序递归的执行划分与合并的过程,当序列中划分到只有一个元素时即递归结束，否则将序列均分为两个等长子序列，在两个子序列中进行归并，最后将两个子序列合并。

（三）排序数据的记录

为了比较各种算法在不同情况下的运行效率，需要在排序过程中记录以下数据：实际运行时间、比较次数、移动次数。为了使代码更为简洁，将三个数据封装为 Info 类，每个算法在排序后返回一个 Info 类实例作为排序数据，由于多个排序算法需要递归调用以及调用辅助函数，所以采取传引用的方式记录信息，也避免了对象的多次复制干扰实际排序时间。

对于比较次数以及元素移动次数只需要通过传引用的方式在排序过程中修改事先定义的 Info 实例的值即可。每一次调用 swap 函数交换元素时使 info.move++。每一次判断 data[i]<=data[j]时使 info.compare++，使用逗号表达式可以在 for 循环里优雅的实现对比较次数的计数。

对于算法运行时间记录，可以利用 ctime 库中的 clock 函数。在此复用了查找性能测试实验中的 Timer 类记录运行时间，具体实现不再赘述。

四、实验结果与分析

实验测试了七种排序算法在规模为 50，500，50000，50000 随机数据下在顺序、逆序、随机情况下的排序结果并汇总到 excel 表格中(见附录)。通过表格可以发现，在数据规模较小的情况下(<=500),各种排序算法的绝对运行时间相差很小，均在 0-3ms 左右，元素的顺序对于运行时间的影响也极小。

随着数据规模的增大，各种算法的绝对运行时间逐渐区分开来。在正序的情况下，插入排序和冒泡排序有着线性阶的效率，在随机和逆序的情况下效率较差，达到了平方阶，这都与两种算法需要大量移动元素有关，在顺序情况下插入排序只需扫描一遍序列、冒泡排序也无序移动元素，效率较高。在逆序和随机的情况下大量的移动导致了两种排序算法效率大大下降。在逆序和随机情况下效率差的还有选择排序，因为其不能识别有序元素，每次都需要进行大量比较，达到了平方阶。在随机的情况下,快速排序的效率最高,为$O(n\log_2 n)$,但在逆序情况下效率最差,这与逆序情况下快速排序在划分序列时除了比较以外,还需要大量移动元素有关,除此以外,递归深度也是影响快排效率的一个因素之一。对于希尔排序、堆排序、归并排序，这三种排序算法的运行效率较为稳定，在正序、逆序、随机的情况下均为$O(n\log_2 n)$左右。

五、小结

通过本次实验，我对各种排序算法的原理有所加深、对于各种排序算法的实现也更加熟练。通过对于不同规模以及情况下各种排序算法的测试和数据分析，我对于各种排序算法有了更加深入的了解与直观的感受。

六、附录

顺序	冒泡排序			快速排序			插入排序			选择排序			归并排序			希尔排序			堆排序		
	移动次数	比较次数	运行时间(ms)	移动次数	比较次数	运行时间(ms)	移动次数	比较次数	运行时间(ms)	移动次数	比较次数	运行时间(ms)	移动次数	比较次数	运行时间(ms)	移动次数	比较次数	运行时间(ms)	移动次数	比较次数	运行时间(ms)
正序	0	1225	0	49	625	0	0	49	0	0	1225	0	286	153	0	0	203	0	270	434	0
逆序	1225	1225	0	25	1225	0	1225	1225	0	25	1225	0	286	133	0	105	263	0	207	379	0
随机	600	1225	0	92	237	0	600	644	0	44	1225	0	286	219	0	158	337	0	250	418	1
正序	0	124750	0	499	62500	1	0	499	0	0	124750	1	4488	2272	1	0	3506	0	4354	7756	1
逆序	124750	124750	3	250	124750	0	124750	124750	1	250	124750	1	4488	2216	1	2100	5116	0	3676	7010	0
随机	62725	125740	2	1599	5013	0	62725	63219	0	489	124750	1	4488	3863	0	3203	6432	0	4025	7392	0
正序	0	12497500	26	4999	6250000	30	0	4999	0	0	12497500	24	61808	32004	4	0	55005	0	60932	112126	2
逆序	12497500	12497500	389	2500	12497500	35	12497500	12497500	47	2500	12497500	31	61808	29804	44	28708	78798	0	53436	103227	2
随机	6295661	12497500	195	24441	69936	1	6295661	6300655	20	4995	12497500	28	61808	55227	12	62403	114768	1	57019	107606	2
正序	0	1249975000	2547	49999	625000000	1886	0	49999	1	0	1249975000	2758	784464	401952	891	0	700006	7	773304	1455438	60
逆序	1249970501	1249975000	81582	25000	1249975000	8259	1249974999	1249975000	9075	25001	1249975000	6734	784464	382512	823	397280	1047305	12	698892	1366047	52
随机	626777336	1249975000	43490	316476	953582	31	626777336	626827326	4524	49990	1249975000	5948	784464	718405	825	1283732	1958258	26	737314	1409715	69

排序算法测试结果