**散列作业上机报告**

1400012141

邵智轩

物理学院

窗体顶端

**一 题目描述**

利用随机数发生器（seed 0），产生不同数量的序列，研究当元素数量上升时，不同排序算法排序所花费的时间（绝对时间ms,移动次数，比较次数等先对时间）；

**二 解决方案**

采用了直接插入排序、二分插入排序、Shell排序、直接选择排序、起泡排序、二路归并排序共6种排序方法。分别研究不同方法下，对数据量为10000、20000、40000排序所花费的时间。对于Shell排序和二路归并排序这两种速度很快的排序，还研究了数据量100000、1000000、10000000排序所花费的时间。

“时间”数据，不仅包含绝对时间ms，还有比较次数和移动次数。此外，对于每一个数据量，我做了乱序、正序、倒序三组测验来研究不同排序的特性。

所得到的数据，附在“排序数据.xls”文件中。

**三 数据结构**

数据表中的每一个元素：

struct RecordNode

{

KeyType key;

DataType info;

}

在本次作业中KeyType即为int类型，DataType用不到。

数据表为顺序表

struct SortObject

{

int max;//最大记录个数

int n;//文件中的记录个数

RecordNode<KeyType, DataType>\* record;

}

文件数据用随机数发生器（seed(0)）生成。

**四 不同排序方法的单独分析**

**4.1 直接插入排序 void insertSort()**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 直接插入排序 | | | | |
| 顺序 | 数据量 | 10000 | 20000 | 40000 |
| 乱序 | 时间 ms | 105 | 420 | 1573 |
| 比较次数 | 25156706 | 100748173 | 399055132 |
| 移动次数 | 25166694 | 100768166 | 399095124 |
| 正序 | 时间 ms | 0 | 0 | 0 |
| 比较次数 | 9999 | 19999 | 39999 |
| 移动次数 | 9999 | 19999 | 39999 |
| 倒序 | 时间 ms | 217 | 801 | 3232 |
| 比较次数 | 50003528 | 200004050 | 799995504 |
| 移动次数 | 50013527 | 200024049 | 800035503 |

可以看出直接插入排序具有如下性质

①排序时间，正序<乱序<倒序。正序耗时很小，而倒序耗时约为乱序的2倍。即课本上所谓的“对于不同文件初态，直接插入排序所耗时间有很大差异。”

②当数据量增加为原来的2倍时，无论是比较次数、移动次数还是绝对时间都大约为原来的4倍，即都为O(n2)。

**4.2 二分插入排序 void binSort()**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 二分插入排序 | | | | |
| 顺序 | 数据量 | 10000 | 20000 | 40000 |
| 乱序 | 时间 ms | 89 | 354 | 1369 |
| 比较次数 | 119011 | 257984 | 555921 |
| 移动次数 | 25166694 | 100768166 | 399095124 |
| 正序 | 时间 ms | 2 | 2 | 4 |
| 比较次数 | 123617 | 267233 | 574465 |
| 移动次数 | 9999 | 19999 | 39999 |
| 倒序 | 时间 ms | 169 | 676 | 2741 |
| 比较次数 | 114189 | 249245 | 541404 |
| 移动次数 | 50013527 | 200024049 | 800035503 |

可以看出二分插入排序具有如下性质

①排序时间，正序<乱序<倒序。正序耗时很小，而倒序耗时约为乱序的2倍。这一点和直接插入排序非常类似。

②对比直接插入排序可以看到：二者在处理相同的数据时，移动次数完全一样。而二分插入排序的优势在于，其比较次数比直接插入排序少得多，在数据量增大时尤其明显。

③当数据量增加为原来的2倍时，移动次数和绝对时间都大约为原来的4倍。即都为O(n2)。而比较次数的增长明显小于这一速度。有教材上的理论分析可知总比较次数。把数据代入计算也验证了这一结果：。由于这相比移动次数的增加忽略不计，故绝对时间仍为O(n2)。

**4.3 Shell排序（缩小增量法） void shellSort(int d)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Shell排序 | | | | | | | |
| 顺序 | 数据量 | 10000 | 20000 | 40000 | 100000 | 1000000 | 10000000 |
| 乱序 | 时间 ms | 3 | 6 | 15 | 39 | 482 | 7153 |
| 比较次数 | 264290 | 623385 | 1448186 | 4157115 | 62839598 | 977475381 |
| 移动次数 | 322056 | 750067 | 1718843 | 4918077 | 71199754 | 1077131149 |
| 正序 | 时间 ms | 1 | 2 | 4 | 10 | 95 | 1188 |
| 比较次数 | 120005 | 260005 | 560005 | 1500006 | 18000007 | 220000008 |
| 移动次数 | 120005 | 260005 | 560005 | 1500006 | 18000007 | 220000008 |
| 倒序 | 时间 ms | 1 | 3 | 5 | 15 | 213 | 1758 |
| 比较次数 | 181958 | 392765 | 840897 | 2273423 | 25899855 | 299779620 |
| 移动次数 | 243383 | 524469 | 1119677 | 3036108 | 33771894 | 379528523 |

可以看出Shell排序具有如下性质

①排序时间，乱序>>倒序>正序。因正序和倒序都是接近于有序状态的顺序。

②在效率上和二路归并排序速度差不多，在同一数量级；比其他四种排序都快得多。

③当数据量从106增加为原来的10倍（即107）时，比较次数为原来的15.6倍，移动次数为原来的15.1倍，绝对时间为原来的14.8倍。都约为，比书上的还要小一些。

**4.4 直接选择排序 void selectSort()**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 直接选择排序 | | | | |
| 顺序 | 数据量 | 10000 | 20000 | 40000 |
| 乱序 | 时间 ms | 181 | 726 | 2900 |
| 比较次数 | 49995000 | 199990000 | 799980000 |
| 移动次数 | 29970 | 59961 | 119967 |
| 正序 | 时间 ms | 180 | 794 | 2921 |
| 比较次数 | 49995000 | 199990000 | 799980000 |
| 移动次数 | 0 | 0 | 0 |
| 倒序 | 时间 ms | 191 | 927 | 4960 |
| 比较次数 | 49995000 | 199990000 | 799980000 |
| 移动次数 | 16707 | 35130 | 74829 |

可以看出直接选择排序具有如下性质

①排序时间，乱序≈正序<倒序。

②在效率上比与直接插入排序和二分插入排序差不多，在同一数量级。

③比较次数与文件初始状态无关。

④当数据量增加为原来的2倍时，比较次数为原来4倍，为；移动次数为原来的2倍，为；绝对时间为原来的4倍，为。

**4.5 起泡排序（交换排序的一种） void bubbleSort()**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 起泡排序 | | | | |
| 顺序 | 数据量 | 10000 | 20000 | 40000 |
| 乱序 | 时间 ms | 561 | 2276 | 9255 |
| 比较次数 | 98210178 | 395820208 | 1592080197 |
| 移动次数 | 75440121 | 302184522 | 1197045399 |
| 正序 | 时间 ms | 0 | 1 | 0 |
| 比较次数 | 9999 | 19999 | 39999 |
| 移动次数 | 0 | 0 | 0 |
| 倒序 | 时间 ms | 556 | 2193 | 8873 |
| 比较次数 | 99980001 | 399960001 | 1599920001 |
| 移动次数 | 149980587 | 599952153 | 2399866515 |

可以看出起泡排序具有如下性质

①排序时间，乱序≈倒序>>正序。正序的耗时几乎为零，因为文件初态为正序时，一趟起泡就可以完成排序。

②就对乱序初态文件的排序效率而言，起泡排序比作业中的其他排序方法都慢很多。

③当数据量增加为原来的2倍时，比较次数、移动次数、绝对时间都为原来4倍，都为。

**4.6 二路归并排序 void mergeSort()**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 二路归并排序 | | | | | | | |
| 顺序 | 数据量 | 10000 | 20000 | 40000 | 100000 | 1000000 | 10000000 |
| 乱序 | 时间 ms | 3 | 6 | 11 | 32 | 290 | 3148 |
| 比较次数 | 123622 | 267429 | 574619 | 1566201 | 18715488 | 224005320 |
| 移动次数 | 140000 | 320000 | 640000 | 1800000 | 20000000 | 240000000 |
| 正序 | 时间 ms | 2 | 4 | 7 | 18 | 185 | 2147 |
| 比较次数 | 71712 | 153424 | 326848 | 877968 | 10095552 | 122213568 |
| 移动次数 | 140000 | 320000 | 640000 | 1800000 | 20000000 | 240000000 |
| 倒序 | 时间 ms | 2 | 3 | 7 | 18 | 190 | 2184 |
| 比较次数 | 65305 | 141894 | 308281 | 861066 | 10776782 | 124280406 |
| 移动次数 | 140000 | 320000 | 640000 | 1800000 | 20000000 | 240000000 |

可以看出二路归并排序有以下性质

①排序时间，乱序>倒序≈正序。因正序和倒序都是接近于有序状态的顺序。

②在效率上和Shell排序速度差不多，在同一数量级；比其他四种排序都快得多。

③移动次数与文件初始状态无关。

③当数据量从106增加为原来的10倍（即107）时，移动次数为原来的12倍，比较次数为原来的11.968倍，绝对时间为原来的10.86倍。书上的理论分析说时间代价为，则当数据量从106增加为原来的10倍（即107）时，时间应增加为原来的12倍，与实际结果很接近。