三种排序算法评价报告

介绍测试工具集

清单

```
./data
./gen.py
./gen.sh
./main.cpp
./Makefile
./std.cpp
./test.sh
```

文件名	功能
Makefile	控制整个测试流程
gen.py	用于生成不同的测试输入
gen.sh	控制测试输入生成以及正确答案的输出
std.cpp	用stl sort实现的标准答案
main.cpp	待测试程序
test.sh	控制测试流程

用法

以下在Ubuntu 18.04中可正常运行，Debian系发行版应该都能够正常运行，WSL上测试亦通过。**需要GNU Make工具、g++以及python2**

`make`自动执行整个测试流程，并输出结果到标准输出

`make binary`编译使用三种不同算法的排序程序，并编译标准程序std

`make std`只编译标准程序std

`make testcase`生成n为100到100000，梯度为10倍，分别具有随机、降序、升序三种特征共12组数据，并调用std生成答案

`make test`调用三种不同算法实现的排序程序对数据进行排序，工具会检查排序结果的正确性并输出腾挪/赋值次数和比较次数

样例输出及注释

`make`

```
100                                         # 此组测试数据大小为100
qsort r AC            304            966    # 快速排序 数据特征为随机 结果正确 腾挪/赋值次数为304,
比较次数为966
msort r AC           1344            547
isort r AC           2792           2785
qsort i AC            297           4950
msort i AC           1344            356    # 归并排序 数据特征为升序
isort i AC            200            199
qsort d AC            297           5000
msort d AC           1344            316
isort d AC           5150           5050    # 插入排序 数据特征为降序


1000
qsort r AC           3765          14420
msort r AC          19952           8680
isort r AC         255907         255900
qsort i AC           2997         499500
msort i AC          19952           5044
isort i AC           2000           1999
qsort d AC           2997         500000
msort d AC          19952           4932
isort d AC         501500         500500


10000
qsort r AC          44992         206306
msort r AC         267232         120428
isort r AC       25148189       25148181
qsort i AC          29997       49995000
msort i AC         267232          69008
isort i AC          20000          19999
qsort d AC          29997       50000000
msort d AC         267232          64608
isort d AC       50015000       50005000


100000
qsort r AC         523928        2705930
msort r AC        3337856        1536236
isort r AC     2493173305     2493173289
qsort i AC         299988     4999851629
msort i AC        3337856         853904
isort i AC         200000         199999
qsort d AC         299987     4999916606
msort d AC        3337856         815027
isort d AC     5000149995     5000050000
```

用于定数循环控制的比较次数并没有被包括在内，统计仅包含了用于决定是否对元素进行操作的那些比较。

结果汇总及分析

https://docs.google.com/spreadsheets/d/1h71QwPLw74WROPpCvlguCgVzClfowX6lOD2USxLbse4/edit?usp=sharing

或见内附的`算法实验.xlsx`

或内附的`table.html`

我们可以看到最基础的**快速排序**的比较次数在随机数据（*平均情况*）下增长速度约为数据每增大10倍，比较次数约增大14倍，腾挪次数约增大12倍。 在升序和降序数据下，腾挪次数/赋值次数增长速度约为10倍，而比较次数则变成了惊人的100倍。而这也正是我们这种快速排序的*最坏情况*，复杂度退化到$O(n^2)$。

我们再看**归并排序**，我们发现两个统计量的增长速率都是13、14的样子，而且相同数据大小的情况下，无论数据具有什么样的特征，赋值次数都是相同的，这是因为无论数据具有什么特征，都会被放入并拿出辅助数组$\log_2 n$遍，我们可以由此给出估算式子$count = 2n \times \log_2 n$。而比较次数在随机数据下没有明显特征，但一定是$O(n \log n)$的。在升序或降序条件（也许这可以算是一种*最好情况*？）下由于每次合并的时候比较总是比较半个区间长度次数就把左面的区间段归位了，右面的直接无脑放入就可以了，所以也应该是$O(n \log n)$的，我们能给出估算式子$count = \frac{1}{2}n \times \log_2 n$。由于归并排序的性质，他的运行时间相当稳定，对于他来说没有什么*最好情况*、*最坏情况*。

最后再来看**插入排序**，我们发现插入排序的比较次数和赋值次数增长速率相同。随机数据（*平均情况*）下均为96，降序条件下均为99，升序条件下均位10。升序条件（*最好情况*）如此与众不同的原因是我们是从后往前插入的，检查的第一个位置就是正确的位置，插入也不需要移动整段序列，所以每次插入都是$O(1)$的，故*最好情况*下复杂度为$O(n)$。

> 这个最好情况跟插排的具体实现息息相关，如果是从前往后找插入位置的话，那么复杂度还是$O(n^2)$

这个版本的插入排序的*最坏情况*就是降序，增长速度达到了99，不过也不是很糟，随机数据的增长速度都有96了呢。

总结

对最终结果的评价要以两个统计量的较大者为标准。

我们发现快速排序和归并排序在平均情况下表现差不多，感觉归并还要更好一点，但实际上由于归并不是原地排序，常数会大一些。需要注意的是，这个快排的实现是最裸的，pivot的选取还有随机选、首中尾取中位数等选法，表现会比这个版本的更好。尤其是遇到有序数组的时候，这些pivot选法将避免复杂度退化到$O(n^2)$。

一个好的排序算法会综合各种排序算法的优点，所以不可以简单的断言快排是最好的排序算法。

后记

为什么不测时？

测时间无外乎一下几种做法

1. 利用shell的built-in的time进行时间测定
2. 利用系统提供的time工具进行时间测定
3. 运行程序前后系统时间作差
4. 利用time.h中的clock进行测时

其中1、2的时间粒度都比较迷，基本上都是10ms的，虽然说足以体现运行的差异，但是会把读入数据的时间也计算在内，不是很好。

方法3的时间粒度可以很细，可以到μSec，但是依然面临会把非核心算法的计算时间也放在里面的这种问题。

至于方法4，方法4不知道到底是谁问题，测定的结果非常迷，经常是0，而且多次测量结果还经常有差别，我觉得是最不靠谱的测定方法。

对于这种输入数据较大的问题，我们所关心的是核心算法的时间复杂度，而不是很关心读入数据输出数据啥的用了多少时间。而我们所写的确定性算法的核心步骤的运行时间往往与赋值、比较操作的次数有一个比较稳定的比例关系，鉴于clock()的不靠谱，我们用这些参数取一个最大值来衡量算法的时间复杂度还是比较靠谱的。最重要的是：同一组数据多次运行，这些参数的结果都是一样的，而不是会和运行时间一样，受到计算机性能波动的影响。如果是比较运行时间增长速度的话，则更加靠谱了，我们比较一下参数的增长速度和数据集大小的增长速度，大概就能估算出这个算法运行时间的增长到底是什么样子的了。