数据结构与算法实验报告 2——排序

软件 52 陈书新 2015013229

csx15@mails.tsinghua.edu.cn

一、实验简介

比较四种不同的排序算法对随机 32 位无符号整数的排序效果，数据范围从 10 至 10^9，在比较中对排序算法进行优化。

二、实验环境

操作系统：macOS Sierra 10.12.3

处理器：2.5 GHz Intel Core i7

内存：16GB 1600MHz DDR3

编辑器：Xcode 8.0 8A218a (Release)

编译环境：clang-800.0.38

三、实验过程

随机数的生成直接使用 c++的 random 库自带的库函数，可以随机生成指定范围内的整数，详情见源代码。

插入排序(Insertion Sort)的时间复杂度为$O(n^2)$，这样显然连 10^6 的数据都跑不过。因此我们必须要进行优化。希尔排序(Shell Sort)是插入排序的一种，它是一种固定间隔的插入排序。已知的最优复杂度的希尔排序由 Pratt 在 1971 年提出，最坏情况下时间复杂度为$O(n\log^2 n)$。平均情况下时间复杂度最优的希尔排序由 SedgeWick 在 1986 年提出，平均时间复杂度为$O(n^{7/6})$，最坏情况下为$O(n^{4/3})$。本实验中我们采用后者。

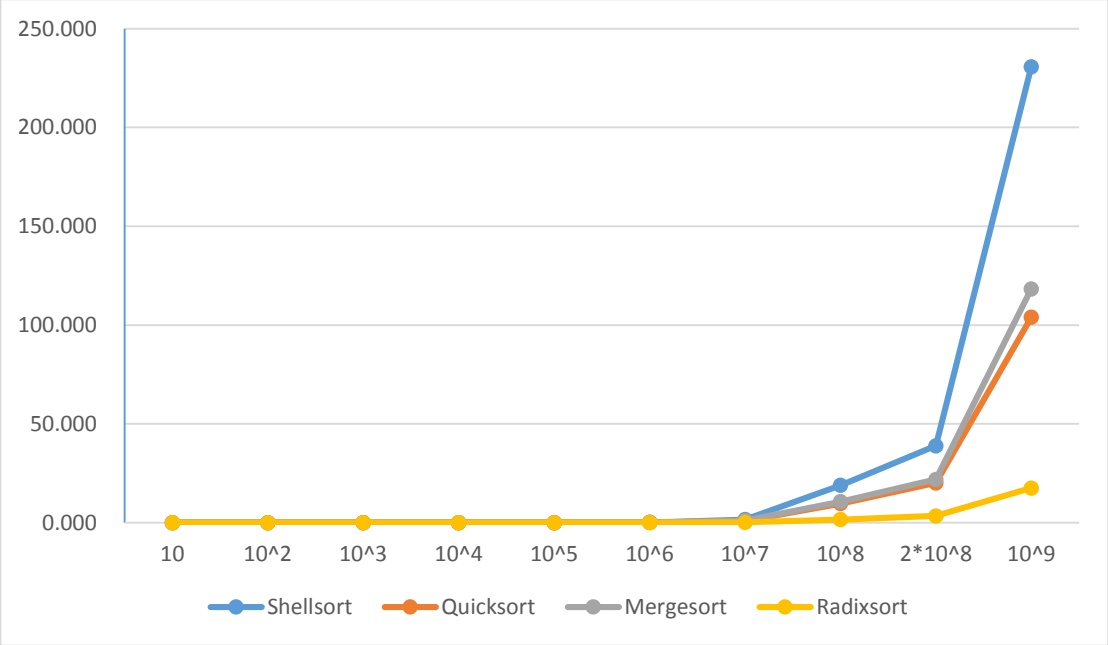
快速排序(Quick Sort)和归并排序(Merge Sort)的时间复杂度均为$O(n\log n)$，但各有优劣。快速排序必须基于递归，但是只需要$O(1)$的辅助空间；归并排序可以用非递归实现，但是需要$O(n\log n)$的辅助空间。

基数排序(Radix Sort)的时间复杂度为$O(n)$，在经过若干次实验后，我们发现每次比较 8 个二进制位时，常数更小。这里需要注意的是，课堂上讲的计算比较二进制位的选择时是用函数求导取极值的方式，但这只是理论的时间复杂度的计算方法。在实际中还要考虑很多硬件层面的情况，比如 Cache 的命中。

四种排序的运行时间见下表。

算法|n	10	10^2	10^3	10^4	10^5
Shellsort	0.000	0.000	0.000	0.002	0.009
Quicksort	0.000	0.000	0.000	0.000	0.006
Mergesort	0.000	0.000	0.000	0.000	0.006
Radixsort	0.000	0.000	0.000	0.000	0.001
算法|n	10^6	10^7	10^8	$2 * 10^8$	10^9
Shellsort	0.119	1.522	18.856	38.891	230.699
Quicksort	0.077	0.863	9.607	20.087	103.955
Mergesort	0.084	0.988	10.647	21.912	118.277
Radixsort	0.012	0.129	1.547	3.385	17.474

根据上表我们得到折线图。



从上表中我们可以发现，实验结果与理论的时间复杂度基本符合一致。在归并排序中，我使用大量的常数优化，例如用指针访问代替数组下标访问，用三目运算符代替 if 等，使得归并排序的速度大约提高了 30%。但由于归并排序需要大量的辅助空间，且每次从辅助空间向目标空间的数据拷贝需要消耗大量的时间，因此仍然要略慢于快速排序。

最后，将其中最快的快速排序与系统自带的 std::sort()进行比较，在不同的 case 中（不仅仅限于无符号 32 位整数），快速排序比系统排序慢 10%~30%，可见常数优化任重而道远。在与 std::stable_sort()进行比较时，快速排序在一些 case 中的运行时间甚至优于系统排序。

四、 参考文献

[1] Cppreference. std::uniform_int_distribution[EB/OL]. http://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution.

[2] Wikipedia. Shellsort[EB/OL]. https://en.wikipedia.org/wiki/Shellsort.