排序算法 实验报告

吴佳龙 2018013418

摘要

本次实验结合理论分析和程序设计，实现了不同的排序算法，具体地，这些方法为：插入排序，希尔排序，快速排序，归并排序，基数排序，并验证了它们的结果正确性，比较了它们的计算时间，实验结果与理论分析相符。最后，还分析了基数排序中分组参数 r 对于计算时间的影响。

1 问题

比较 Insertion Sort, Shell Sort, Quick Sort, Merge Sort, Radix Sort 对 32 位无符号整数的排序效果。输入数据随机产生，数据范围为 $[0, 2^{32})$, 输入数据量分别为

$$10, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8, 2 \times 10^8$$

2 实验环境

操作系统：Windows 10
IDE：Visual Studio 2017
处理器：3.1 GHz 双核 Intel Core i5

3 算法分析

本次实验共 5 种算法，分别在以下 5 个小节中进行算法描述与分析。

3.1 插入排序

```
void InsertionSort(value_t *a, int n);
```

算法描述 对 $i = 1 \to n$ 依次将 a_i 插入到已经排好序的 $a_{1\cdots i-1}$ 中的合适位置。

时间复杂度分析 每次插入最坏情况下的交换次数为 $\Theta(i)$，最坏的总时间复杂度为 $\Theta(n^2)$；最好情况下交换次数为 $\Theta(1)$，最好的总时间复杂度为 $\Theta(n)$

空间复杂度分析 不需要辅助空间。

3.2 希尔排序

```
void ShellInsert(value_t *a, int n, int d);
void ShellSort(value_t *a, int n);
```

算法描述 对一组单调下降到 1 的增量数组 d_i 依次进行 ShellInsert，这一过程将下标间隔为 d_i 的子序列进行插入排序，共有 d_i 组这样的子序列被进行插入排序。

增量的取法没有统一的约定，本次实现中，取

$$d_0 = n, d_i = \lfloor d_{i-1}/2 \rfloor + [\lfloor d_{i-1}/2 \rfloor \bmod 2 = 0]$$

其中 $[\cdot]$ 表示艾弗森约定，当其中的表达式为真时值为 1，否则为 0。也就是说在这里我们要求 d_i 是奇数。注意这仅是一种增量数组的取法。

时间复杂度分析 由于被进行大的增量的插入排序之后，数组变得“近似有序”，因此之后排序的交换次数会有所减少。希尔排序的交换和比较次数大约在 $O(n^{1.3})$

空间复杂度分析 不需要辅助空间。

3.3 快速排序

```
int Partition(value_t *a, int l, int r);
void QuickSort(value_t *a, int l, int r);
```

算法描述 每次递归的过程中，选择一个主元，将小于等于主元的元素放到当前区间的左半边，其余放到右半边，对两边分别递归进行。

时间复杂度分析 最坏的情况下，主元最终位于区间的两端，$T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$；最好的情况下主元最终位于区间的中间 $T(n) = 2T(n) + \Theta(n) = \Theta(n \lg n)$；在数据随机的情况下，期望的复杂度 $O(n \lg n)$

空间复杂度分析 不需要辅助空间。

3.4 归并排序

```
void MergeSort(value_t *a, int l, int r,
    value_t *tmp);
```

算法描述 每次递归过程中，分别对左半边和右半边递归地排序，之后将两个有序的子列合并在一起。

时间复杂度分析 合并的过程的复杂度为 $\Theta(n)$，因此总的复杂度

$$T(n) = 2T(n) + \Theta(n) = \Theta(n \lg n)$$

空间复杂度分析 合并的过程中需要 $\Theta(n)$ 的辅助空间。

3.5 基数排序

```
void RadixSort(value_t *A, int n, int w, int
    r, value_t *tmp, int *c);
```

算法描述 对每个元素的共 w 位，从低位到高位每 r 位分成一组。从低位到高位，对整个数组依次根据每一组二进制位上的值进行稳定的排序（一般选择计数排序），供需进行 $\lceil \frac{w}{r} \rceil$ 次计数排序，最终得到有序的数组。

时间复杂度分析 一次计数排序的复杂度为 $\Theta(n + 2^r)$，总的复杂度为

$$\Theta(\frac{w}{r}(n + 2^r))$$

理论上，当 $r = \lg n$ 时该函数取到最优值

$$\Theta(\frac{wn}{\lg n})$$

空间复杂度分析 需要 $\Theta(n + 2^r)$ 的空间用作计数排序

参数 r 的选取 在本次实验中固定 $w = 32$，虽然理论上有最优的 r 的选取（具体实现中我们取 $r = \lceil \lg n \rceil$），我们还固定了 r 的两个值 $r = 11$ 和 $r = 16$ 分别观察了排序效率，这两个值的选取是因为他们分别能在 3 次和 2 次计数排序之内完成基数排序。

4 结果分析

4.1 结果正确性

首先我们验证了算法实现的正确性，验证方法为：随机生成指定规模的数组 a_i，并调用不同算法对它进行排序，再两两比较不同算法的结果。

实验表明，对于随机生成的数组，所有算法的排序结果总是相同的。由此可以认为我们对于算法的实现是无误的。

随机数的产生 假设 $rand15()$ 能够产生均匀的无符号 15 位随机整数（在 Windows 下的 C++ 语言中，对应标准库中的 `rand()` 函数），则

$$rand15() \times 2^{17} + rand15() \times 2^2 + rand15() \bmod 2^2$$

能够产生均匀的无符号 32 位随机整数。

4.2 计算时间

对题中要求的不同规模的数据量，随机生成数组 $a_i \in [0, 2^{32})$，调用不同的算法，并多次运行取平均排序时间，得到的计算时间如表 1，图 2，计算时间的自然对数如图 1 。

部分数据由于计算时间过长（插入排序 $n > 10^6$）或者实验环境内存不足（基数排序 $r = \lg n, n = 2 \times 10^8$）未能测出。

n	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8	2×10^8
Insertion	1.521×10^{-7}	1.998×10^{-6}	1.194×10^{-4}	0.015	1.209	115.366			
Shell	2.117×10^{-7}	4.131×10^{-6}	6.737×10^{-5}	0.001	0.014	0.187	2.291	29.630	63.882
Quick	3.010×10^{-7}	3.883×10^{-6}	4.716×10^{-5}	5.780×10^{-4}	0.007	0.080	0.930	10.628	21.825
Merge	3.572×10^{-7}	5.778×10^{-6}	7.283×10^{-5}	8.739×10^{-4}	0.011	0.131	1.526	17.385	36.185
Radix_{11}	1.066×10^{-5}	1.150×10^{-5}	2.094×10^{-5}	1.670×10^{-4}	0.001	0.014	0.153	1.649	3.439
Radix_{16}	2.321×10^{-4}	2.292×10^{-4}	2.429×10^{-4}	3.420×10^{-4}	0.001	0.019	0.210	2.664	6.676
$\text{Radix}_{\lg n}$	7.045×10^{-7}	2.729×10^{-6}	1.951×10^{-5}	2.345×10^{-4}	0.002	0.043	1.458	17.745	

Table 1: 不同算法在不同输入规模下的计算时间

结果分析 当 n 较大时，插入排序的排序时间最长，增长最快，希尔排序次之，这与他们多项式级别的时间复杂度（分别为 n^2 和 $n^{1.3}$）相一致。快速排序与归并排序虽然时间复杂度相同，但是快速排序缓存命中率更高，因此快速排序理论上和实验上都快于归并排序。关于基数排序的排序时间分析详见下一小节。

我们还观察到，在横纵坐标都取对数的情况下，所有排序的时间增长都成线性，与他们的时间复杂度相符合，这是因为

$$\ln(n \lg n) = \ln n + \ln \lg n$$

$$\ln(n^\alpha) = \alpha \ln n$$

而其中的 $\ln \lg n$ 一项增长极其缓慢。

另外，当 n 很小时，插入排序的效率最高。

4.2.1 基数排序时间分析

$r = \lg n$ 的基数排序当 n 较小时，快于归并排序和快速排序，这与其时间复杂度相符，但是随着 n 的增大，$r = \lg n$ 的基数排序的时间逐渐与归并排序的时间重合，慢于快速排序。实际上，当 $r = \lg n > 16$ 时，随着 n 的增加继续增加 r 是没有意义的，因为总是需要两次计数排序来完成基数排序。

$r = 11$ 和 $r = 16$ 的基数排序，虽然 n 较小时较慢（这是由于复杂度中的 2^r 中一项导致的），但是当 n 特别大（$n \approx 10^8$），明显快于归并排序、快速排序以及 $r = \lg n$ 基数排序。而其中 $r = 11$ 快于 $r = 16$，这可能与计数排序的次数相矛盾。实际上，根据我有限的计算机组成原理的知识，这可能是因为 2^{11} 大小的数组能够非常有效的利用 L3 级别的高速缓存（大小约在 1-10MB 这一数量级）。

5 总结：不同方法的比较

插入排序 实现简单，且在 n 很小时特别快，因此在 TimSort 等混合的排序算法中经常用于处理规模小的子问题。但是随着 n 增长，排序时间快速增长。

希尔排序 实现也十分简单，且当 n 增长时比插入排序增长地慢得多。

快速排序 $O(n \lg n)$ 的期望复杂度以及较高的缓存利用率，在排序元素进行随机化处理后，特别适合常见的排序场景。这种递归分治的思想还在计算几何中被借鉴，产生了 QuickHull 快速凸包算法计算凸包。

归并排序 $\Theta(n \lg n)$ 的时间复杂度不随数据好坏而变化，且可以在工程上继续优化为 TimSort 等现实中更高效的算法。

基数排序 不基于比较的算法，具有线性的时间复杂度。由于浮点数二进制存储的设计，基数排序也可对计算机存储的浮点数进行排序。在实际场景中，对于 2^{32} 范围内的数据，由于高速缓存的利用，分三段排序 $r = 11$ 要优于分两段排序 $r = 16$。

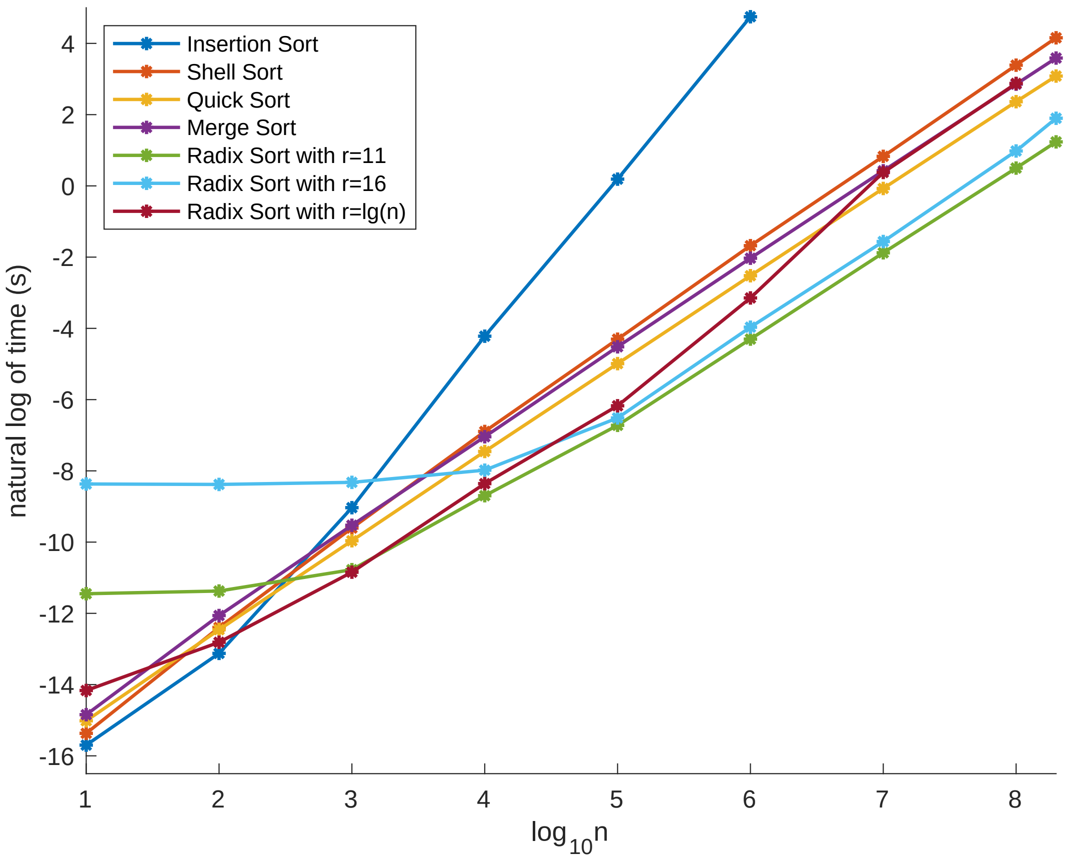

Figure 1: 不同算法在不同规模下的计算时间的自然对数

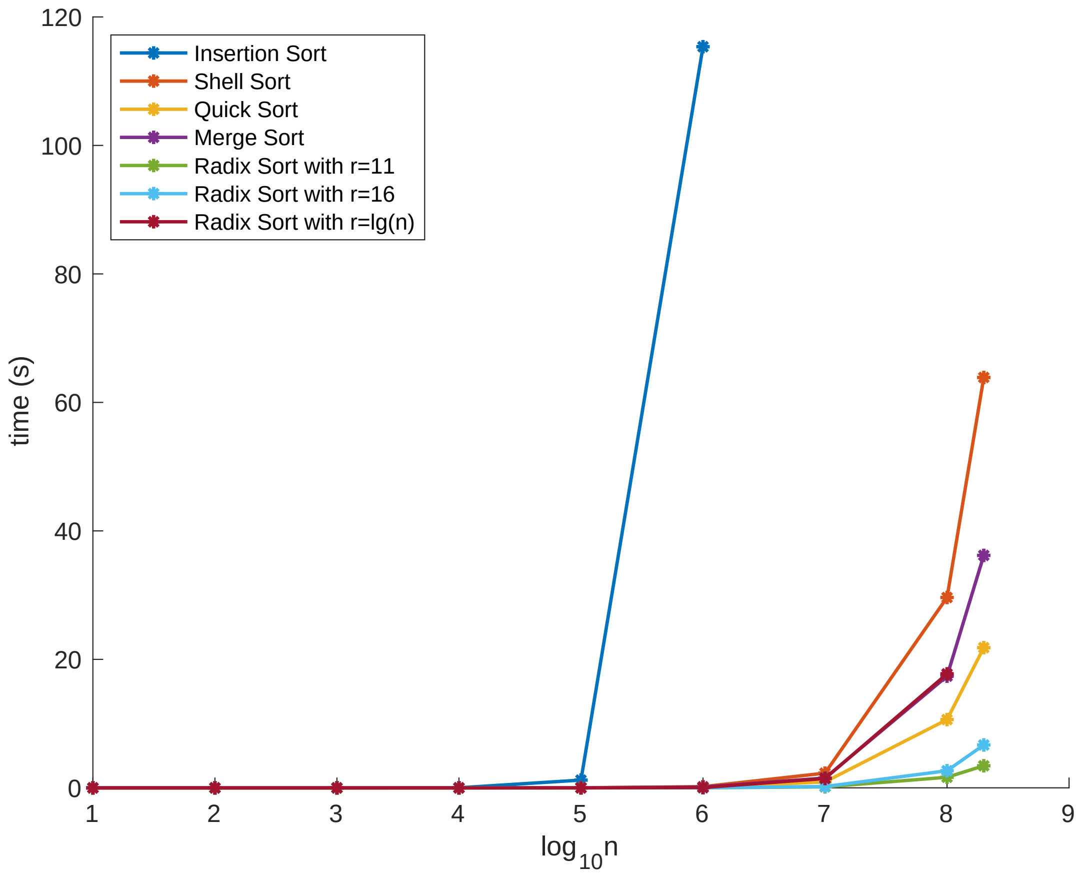

Figure 2: 不同算法在不同规模下的计算时间