对快速排序算法的一些分析

清华大学计算机系，张晨，2017011307

摘　要　本文分析了快速排序的期望和方差，考察了三数取中法、用插入排序代替、中间数法、荷兰国旗法、去重法等快速排序优化方法的实际表现。分析与实验表明，三数取中、插入排序代替两优化结合可有效提高一般情况下快速排序的运行速度，其它各类优化算法在特定的输入数据下也有十分精彩的表现。

关键词　快速排序，插入排序，三数取中，荷兰国旗问题，数据去重

1　引言

排序是数据处理中最基础、最重要的算法之一，不难证明，基于比较的排序的复杂度下界是$O(nlogn)$，故寻找到一个常数尽可能小的$O(nlogn)$ 的排序算法，成为排序问题研究的主要问题。目前，较为经典的时间复杂度为$nlogn$的排序算法有快速排序、归并排序、堆排序、希尔排序等。快速排序是以上算法中最高效的一种，但是，在最坏情况下，其时间复杂度会退化至$O(n^2)$。本文分析了快速排序在两种特殊数列上的运行效率，并尝试对快速排序算法进行优化[1] 。

论文剩余部分如下。第二部分分析了输入序列中元素互异时，随机快速排序算法的比较次数的期望及方差。第三部分提出了3种快速排序的优化方法，并分析了它们在输入数据互异时的表现。第四部分研究了序列中重复元素较多时的快速排序算法。第五部分是对本文的总结。

2　随机快速排序算法

快速排序算法使用了分治的思想，对数组$A[l..r]$的排序可分为以下三步：

主元选取 从$A[l..r]$中任选一个数$x = A[i], i \in [l, r], i \in Z$ 作为主元。

划分 将数组$A[l..r]$分成$A[l..q-1]$,$A[q]$和$A[q+1..n]$三段，使得$A[l..q-1]$中的每一个元素都不大于x，$A[q]$等于x，$A[q+1, r]$中的每一个元素都不小于x。

递归 递归对$A[l..p-1]$、$A[p+1..r]$进行排序，直到$l \geq r$。

下面对随机快速排序的表现进行分析

2.1　期望分析

维基百科[1]证明了在输入数据为一个“随机排列”（n个不同元素组成的数列的$n!$个排列以相等概率出现）时，随机快速排序的期望比较次数为

$$E_n = 1.39n \log_2 n$$

证明：对于一个长度为n的随机排列，被选为主值的数在这个排列中的排名服从1到n的均匀分布，因而“划分”出的三段的大小为i,1,n-i-1，其中i服从0到n-1的均匀分布，划分的过程需要n-1次比较。故对长度为n的随机排列进行随机快速排序的期望比较次数$C(n)$满足

$$C(n) = n - 1 + \frac{1}{n}\sum_{i=0}^{n-1}(C(i) + C(n-i-1))$$

[1]本文涉及的代码均由c++实现，编译器版本为g++ (tdm-1) 5.1.0，可以在https://github.com/heheda12345/sort_analyze下载源代码及原始数据。

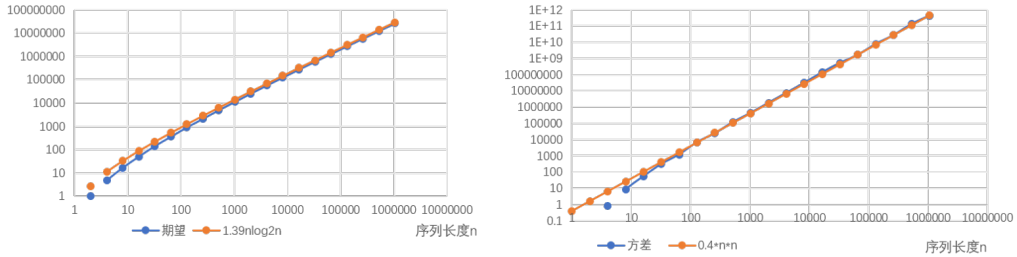

Figure 1: 随机排列下随机快排比较次数的期望与方差，采用2.2节的测试方法进行测试。可发现，其与$E_n = 1.39nlog_2n$，$\sigma_n \approx 0.4n^2$大致吻合。

化简得$C(n) \approx 2\ln n \approx 1.39n\log_2 n$。
从图1中，我们也可以得到相似的结论。

2.2 方差分析

直接推导随机快排方差的通项公式十分困难，因此我通过模拟法估计方差的近似值。

对于$n \in \{2^k \mid 0 \leq k \leq 20, k \in Z\}$，我分别生成了100个长度为n的随机排列，记录对它们进行排序的比较次数$c_1, c_2, ..., c_{100}$。使用公式

$$E_n = \frac{1}{100} * \sum_{i=1}^{100} c_i$$

、

$$\sigma_n^2 = \frac{1}{99} * \sum_{i=1}^{100} (c_i - E_n)^2$$

估计比较次数的期望和方差。计算结果如图1所示。对n与σ_n^2/n进行直线拟合，得到

$$\sigma_n^2/n = 0.4062n + 1319.7$$

相关系数

$$r = 0.996$$

在n较大时，常数项可忽略不计，因此可认为

$$\sigma_n^2 \approx 0.4n^2$$

。

2.3 局限性

尽管快速排序期望时间复杂度十分优秀，但是，其实际时间复杂度却十分依赖于主值的选取。若每次都“幸运”地选取了中位数作为主值，则其比较次数$C(n) = n-1+2*C((n-1)/2) \approx nlog_2n$，而若每次都”不幸“地选择了最大值或最小值作为主值，则比较次数会退化到$C(n) = n-1+C(n-1) \approx \frac{n^2}{2}$。需要特别注意的是，为提高运算效率，很多人在实现随机快速排序时，会选择将最左边或最右边的元素作为主值。如此实现的快速排序，在输入数据为一个基本有序的序列时，会有十分糟糕的表现（例如，若输入数据已经有序，则每次选择的主值都为序列中的最大值或最小值，按之前的分析，比较次数约为$\frac{n^2}{2}$），而输入序列基本有序，是数据处理中很容易出现的情况。因而，我们必须对快速排序进行进一步优化。

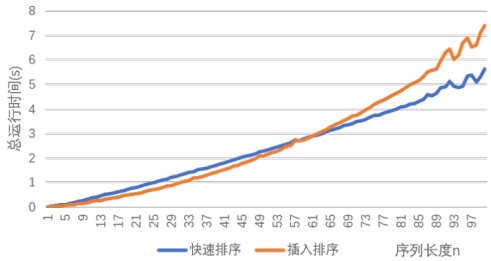

Figure 2: 快速排序与插入排序运行时间的比较。实验方法为对$1 \leq n \leq 100$，随机生成10^6个长度为n的序列，记录对这些排序所用总时间。

3 快速排序算法的优化

本节提出了三种快速排序算法的优化方法。为简化分析过程，本节中的输入数据都为1至n 的随机排列。

3.1 与插入排序结合

虽然插入排序的时间复杂度为$O(n^2)$，但是其不涉及递归，故在数据量n较小的时候，运行速度快于快速排序。图2为快速排序、插入排序在对10^6 个长度为n的随机排列进行排序所花费的时间总和的统计。按照我的实现方式，当$n <= 60$时，插入排序的速度快于快速排序，当$n > 60$时，插入排序的速度则慢于快速排序。

3.2 三数取中

为了使主值尽可能地靠近中间值，可以采用选取待排序的数列中任取的三个数的中位数作为主值。加入这一优化后的期望比较次数为

$$
\begin{aligned}
E_n &= n - 1 + k + \frac{1}{C_n^3}(\sum_{i=0}^{n-1}(E_i + E_{n-i-1}) * i * (n - i - 1) \\
&= n - 1 + k + \frac{2}{C_n^3}(\sum_{i=0}^{n-1} E_i * i * (n - i + 1)) \\
&= n - 1 + k + \frac{2}{C_n^3}((n-1)\sum_{i=0}^{n-1} E_i * i - \sum_{i=0}^{n-1} E_i * i^2)
\end{aligned}
$$

其中，k表示找到三个数的中位数的期望比较次数，在我的实现中，$k = 3/8$。

期望比较次数的通项公式较难计算，因此我使用计算机进行递推计算。我计算了E_n的前10^6项。将n与$\frac{E_n}{\log_2 n}$进行直线拟合，得到$\frac{E_n}{\log_2 n} = 1.1193n - 1974.7$，相关系数$r \approx 1$，因此可估计$E_n = 1.12nlog_2n$。

在实现三数取中时，我选取了最左、中间、最右的三个数的中位数作为主值，对于随机排列，这样选取与任取三个数没有本质区别。实测发现，在输入数据量较大的情况下，三数取中可以有效地降低比较次数与运行时间，但是在数据量较小的情况下，三数取中反而会使程序运行效率变低。这是因为三数取中这一过程需要进行一些额外的比较（按我的实现方式，三数取中所需比较次数的期望值为8/3）。在数据量较大的情况下，三数取中的消耗可以忽略不计，但是在数据量较小的时候，三数

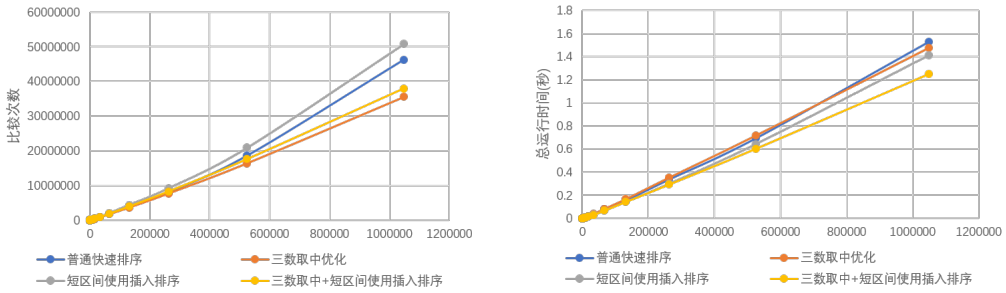

Figure 3: 四种快速排序实现方法的对比。实验方法为对每个n，随机生成10个长度为n的序列，记录排序所需要的平均比较次数、10个序列的总运行时间。四种方法分别为：(1)普通快速排序(2)加入三数取中优化(3)对长度不大于60的区间使用插入排序(4)同时加入优化(2)(3)。

取中操作甚至会变成比较次数的最主要来源。所幸，在数据量较小的时候，可以用插入排序替代快速排序。在进行了这一替换后，三数取中的排序较原有排序算法有了较为明显的优化。测试结果如图3所示。

三数取中排序的另一好处是，在输入数据为一个基本有序的序列时，复杂度为$\Theta(n \log n)$，而不会退化到$\Theta(n^2)$。但是，三数取中排序并没有完全解决快速排序有概率退化到$\Theta(n^2)$的问题，例如，若每次选取的三个数都包含序列中的最小数与次小数，或是最大数与次大数，比较次数为$C_n = n - 1 + \frac{8}{3} + C_{n-2} \approx \frac{n^2}{4}$。

3.3 选取中间数

另一种优化思路是，选取数列中最大值与最小值的平均值作为主值。对于1-n的排列，此算法可保证选取到最理想的主值。若在递归过程中维护最大、最小值，可将期望比较次数降低至$nlog_2n$（不过，1-n的排列存在$O(n)$的排序算法）。但是，对于更一般的序列，求解最大、最小值意味着引入更多的比较次数。我所知最优的求解序列最大、最小值的算法需要$\lceil 3n/2 \rceil - 2$次比较[2]，因此会使最优状态下的比较次数增加至$2.5nlog_2n$。如果数据分布不均匀，此方法无法保证两区间长度近似相等，会使比较次数进一步增加。

不过，如果求序列准确的最大值、最小值，而是根据输入数据的分布对最大值、最小值进行估计，这个算法还是有用武之地的。

例如，如果输入数据为均匀分布，可认为递归时的两个子区间的值域分别为$[min, mid]$，$[mid, max]$，其中，min、max表示父区间的最大、最小值，$mid = (min + max)/2$为父区间的主值。

Algorithm 1 区间快排

```
1:  function RangeSort(Array, left, right, min, max)
2:      L ← left
3:      R ← right
4:      if max − min ≤ 1 then return
5:      end if
6:      pivot ← (min + max)/2
7:      while L ≠ R do
8:          if Array[L] < pivot then
9:              L ← L + 1
10:         else
11:             R ← R − 1
```

```
        swap Array[L] and Array[R]
    end if
  end while
  if L ≠ left then
    RangeSort(Array, left, L, min, mid)
  end if
  if right ≠ R then
    RangeSort(Array, right, R, mid, max)
  end if
end function
```

为了实现的简洁性，该伪代码中的待排序区间为$Array[left, right-1]$，猜测的值域为$[min, max-1]$。此外，该算法在待排序区间较短的情况下表现不是特别理想，因此也可以选择在待排序区间较短时采用插入排序。该算法的实际表现如图4所示。可以发现，该算法大大减少了快速排序所需的比较次数。

	快速排序	区间快排	优化快速排序	优化区间快排
1	0	0	0	0
2	1	2	1.7	1.7
4	5.7	8	6.9	6.9
8	18.5	24	22.1	22.1
16	52.3	64	71.5	71.5
32	143.9	160	262.2	262.2
64	388.5	384	714.6	610
128	907.1	896	1706.5	1551
256	2118.5	2048	3925.6	3457.7
512	5110.8	4608	8560.2	7612.4
1024	11532.7	10240	18885.8	16087.7
2048	25191	22528	41790.4	34137.4
4096	56895	49152	89212.8	72184.5
8192	124359	106496	191336	152594
16384	272677	229376	411263	322527
32768	605382	491520	880633	676978
65536	1263010	1048580	1860310	1420420
131072	2735560	2228220	3926870	2970140
262144	5838390	4718590	8301740	6198160
524288	12493800	9961470	17579000	12917200
1048576	26768800	20971500	36798600	26912700

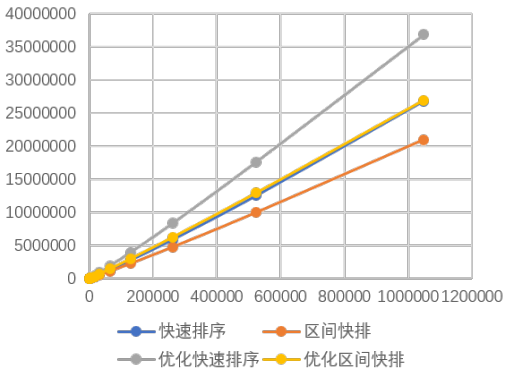

Figure 4: 快速排序与区间排序平均运行次数的对比。优化版快速排序加入三数取中与短区间插入排序两优化，优化版区间排序加入了短区间插入排序一个优化。测试方法为随机生成十个0至n-1(n为序列长度)的序列，求各个排序算法对这个序列进行排序的平均比较次数。

4 数据重复率较高时的排序算法

若待排序序列中有很多重复元素，快速排序会有很糟糕的表现。此问题的极限情况是，若序列中所有元素都相等，则在每次划分后，左半部分是空的，右半部分只减少了一个（因为所有元素都不小于主值）。为解决这一问题，需要使用新的算法。

4.1 荷兰国旗问题

此问题[3]由Edsger Dijkstra提出。荷兰国旗是由红、白、蓝三色组成的。给定随机排成一条直线的这三种颜色的小球，要求高效地将这些球按照红球、白球、蓝球排列。

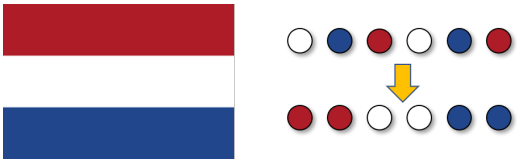

Figure 5: 荷兰国旗与荷兰国旗问题

排好序的序列将会分成前、中、后三个部分，每个小球（红白蓝分别对应0,1,2）恰好属于其中一个。算法总思路是分别构造出序列的前部和后部，则剩下的部分便是序列的中部。使用两个指针start和end指示前部和后部的边界，初始状态下分别指向序列的开头和结尾，再使用一个指针cur表示目前扫描到的位置，初始位于序列开头。

1. 若 cur 指向一个 0 ，则它属于前部。应把它与 start 所指的元素交换，令 cur 、 start 各前进一个位置。

2. 若 cur 指向一个 1 ，则它属于中部，因此不需要交换操作，直接令 cur 前进一个位置。

3. 若 cur 指向一个 2 ，则它属于后部。应把它与 end 所指的元素交换，并令 end 后退一个位置。交换完毕后 cur 可能指向一个 0 ，此时前进 cur 会导致这个 0 无法被交换到前部，故不前进 cur 。

当cur指向end后的第一个元素时，算法运行结束。

令红球、白球、蓝球分别表示“小于主值”、“等于主值”，“大于主值”，则这个算法即为快速排序的“划分”操作。完成划分后，中部已经排好序，只需递归对前部、后部排序。

加入“荷兰国旗”优化的快速排序问题的期望比较次数可用均摊法分析。每次将 cur 指向的元素与主值key比较后，若$cur \leq key$，则 cur 会前进一个位置，若$cur > key$，则 end 会后退一个位置。当 cur 移动到 end 之后时，划分结束。因此整个划分过程需要的比较次数恰好等于区间长度。若在划分前，主值位于序列最左侧，则可令start指向序列开头，cur指向序列中第二个元素，此时只需要区间长度-1次比较，与原本的划分操作所需要的比较次数相等。因此，该优化不会使快速排序算法变坏。

4.2 去重

如果在排序前明确知道序列中元素重复现象较为严重，则可以在排序前对序列进行去重。若序列中的元素的可能取值不多，可使用 bitmap 完成此项工作，否则可选用哈希表。这二者的时间复杂度都为$O(n)$，n表示序列长度，但哈希表的常数较大。

5 总结

本文分析了快速排序的期望和方差，并提出了多种快速排序的优化方法。三数取中选取主值，在区间长度较小时使用插入排序，有效地提高了快速排序的运行速度。此外，本文对输入序列中重复元素较多的情况提出了优化方案。但是，这些算法都没有从根本上解决快速排序时间复杂度可能退化至n^2的问题。因此，实现快速排序时，会选择当快速排序执行时间过长时，换用其它常数略大但时间复杂度为严格$O(nlogn)$的排序算法。

References

[1] Wikipedia contributors, "Quicksort", *Wikipedia*, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 9 Dec. 2018. Web. 9 Dec. 2018.

[2] 邓俊辉,数据结构习题解析, pp. 43,清华大学出版社,2018.1

[3] Wikipedia contributors. "Dutch national flag problem." *Wikipedia*, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 10 Oct. 2018. Web. 10 Dec. 2018.