Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 369 (2006) 44–66
Theoretical Computer Science
www.elsevier.com/locate/tcs

Efficient sample sort and the average case analysis of PEsort☆

Jing-Chao Chen

School of informatics, DongHua University, 1882 Yan-an West Road, Shanghai 200051, PR China

Received 17 July 2005; received in revised form 23 February 2006; accepted 18 July 2006

Communicated by J. Díaz

Abstract

The purpose of the paper is twofold. First, we want to search for a more efficient sample sort. Secondly, by analyzing a variant of Samplesort, we want to settle an open problem: the average case analysis of Proportion Extend Sort (PEsort for short). An efficient variant of Samplesort given in the paper is called *full sample sort*. This algorithm is simple. It has a shorter object code and is almost as fast as PEsort. Theoretically, we show that full sample sort with a linear sampling size performs at most $n \log n + O(n)$ comparisons and $O(n \log n)$ exchanges on the average, but $O(n \log^2 n)$ comparisons in the worst case. This is an improvement on the original Samplesort by Frazer and McKellar, which requires $n \log n + O(n \log \log n)$ comparisons on the average and $O(n^2)$ comparisons in the worst case. On the other hand, we use the same analyzing approach to show that PEsort, with any $p > 0$, performs also at most $n \log n + O(n)$ comparisons on the average. Notice, Cole and Kandathil analyzed only the case $p = 1$ of PEsort. For any $p > 0$, they did not. Namely, their approach is suitable only for a special case such as $p = 1$, while our approach is suitable for the generalized case.

Keywords: Sorting; Samplesort; Complexity of algorithms; Master theorem; Quicksort; Proportion extend sort

1. Introduction

Proportion Extend Sort (PEsort for short) [4] was recently proposed a sorting algorithm with Quicksort flavor. Because it is simple and efficient, and performs $\Theta(n \log n)$ comparisons in the worst case. Soon, Chen [5] made use of it to develop the library sort function. Empirical results showed its library version is efficient, and has weak adaptivity and excellent caching behavior, and can compete with Bentley and McIlroy's Quicksort (BM qsort for short) [1], which is the fastest currently known derivative of Quicksort [14]. In [4], the average case analysis of PEsort was put forward as an open problem. Based on empirical studies, Chen conjectured that PEsort requires $n \log n + O(n)$ comparisons for any $p > 0$, where p is a performance parameter of PEsort. All the log-notations mentioned throughout this paper are a logarithm that is taken over base two, unless otherwise mentioned.

Cole and Kandathil [7] used partition sort to present the average case analysis of PEsort with $p = 1$, and pointed out that it performs at most $n \log n + (\mu - 1.193)n + O(\log n)$ $(0 \leqslant \mu < 0.086)$ comparisons on the average. However, they did not analyze the case of any $p > 0$. In practice, a relatively efficient case is $p = 16$, not $p = 1$. By careful observation, we noted that their analysis approach is suitable only for a special case such as $p = 1$, not for the generalized case.

☆ This work was partially supported by the National Natural Science Foundation of China Grant 60473013.
E-mail address: chen-jc@dhu.edu.cn.

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.07.017

In order to analyze the generalized case of PEsort, we introduce a variant of Samplesort for sorting a partially sorted array. We show that the comparison cost of the variant Samplesort upper bounds that of PEsort. Using this and Roura's Master Theorem [24], we derive that the average case number of comparisons required by PEsort is at most

$$n\log n + \left(\left(\frac{(1+p)(2\ln 2-1)\log(1+p)}{p} + \mu_1\right)\beta - 2\ln\beta - 2 + \mu_2\right)n + \mathrm{o}(n)$$

where $0 \leqslant \mu_1,\ \mu_2 < 0.0861,\ 1/(p(1+p)) < \beta \leqslant 1/p$ when $p \geqslant 1$, and $1/(1+p) < \beta \leqslant 1$ when $1 > p > 0$. When $p = 1$, the above formula is equal to $n\log n - Cn + \mathrm{o}(n)$ $(0.098 \leqslant C \leqslant 1.227)$, which coincides with Cole and Kandathil's result as shown above. One will see that our analysis approach is more rigorous than their analysis approach.

Samplesort is used not only for analyzing the average case of PEsort, but also for developing an efficient sorting algorithm. Based on this feature, we devise a practical variant of Samplesort, which is christened *full sample sort*. The basic principle of the algorithm is the same as Samplesort, but sorts each sample and each bucket recursively. The theoretical analysis shows that when the sample size is fixed, it makes $\Theta(n^2)$ comparisons at most and $Cn\log n + \mathrm{O}(n)$ comparisons on average, where $C > 1$. When the sample size is linear in n, it makes $\Theta(n\log^2 n)$ comparisons at most and $n\log n + \mathrm{O}(n)$ comparisons on average. Notice, it has been shown that PEsort performs $\Theta(n\log^2 n)$ exchanges in the worst case [7]. Theoretically, full sample sort with linear-size samples is more efficient than any known variant of Samplesort. In practice, some versions of it are efficient. For example, when sample size $s = n/24$, the algorithm is as fast as PEsort. In details, we show that the expected number of comparisons when $s = n/24$ is no more than $n\log n - 0.066n + \mathrm{o}(n)$ approaching the information theoretic lower bound for comparison-based sequential sorting algorithms, and the number of exchanges is not also large, less than $0.272n\log n + \mathrm{O}(n)$. Applying an equal-space sampling technique to it, we can easily avoid extreme slowdown $((\Theta(n\log^2 n))$ time) on plausible inputs. Furthermore, by a preprocessing, we can get a optimized version with *Rem*-adaptivity (its definition will appear below), which runs in $\mathrm{O}(n\log n)$ time for random inputs, and in $\mathrm{O}(n)$ time for nearly sorted inputs. The object code of the library function based on the algorithm is shorter than both Psort [5] and BM qsort.

2. Prior work

At first Samplesort was proposed as a sequential sorting algorithm by Frazer and McKellar [10]. Later it was widely used in parallel sorting algorithms [2,9,12,25]. Moreover, there has been a success in the aspect of parallel algorithms. Especially for larger data sets, it can outperform other parallel sorting algorithms. However, no successful example has been seen in the aspect of sequential algorithms. In theory, Samplesort is efficient on average. By Frazer and McKellar's analysis [10], it makes an expected $n\log n + \mathrm{O}(n\log\log n)$ number of comparisons. However, in practice, Samplesort is not so efficient since the implementation way of Frazer and McKellar is fairly complex. Also, in the worst case, Samplesort can go quadratic time. Rajasekaran and Reif [23] made efforts to improve the expected time of Samplesort. Theoretically, the expected number of comparisons can be bounded by $n\log n + \mathrm{O}(n\omega(n))$ [23], for any function $\omega(n)$ (say $\omega(n) = \log\log\log n$) that tends to infinity. The improvement makes Rajasekaran and Reif's algorithm more complicated than Frazer and McKellar's one. It is hard to efficiently implement it. Therefore, Rajasekaran and Reif's improvement is of only theoretical interest, not practical interest. For analysis purpose, Cole and Kandathil [7] presented another Samplesort, called Partition sort, which is almost the same as Frazer and McKellar's Samplesort, except for using insertion sort to sort each subsequence of size $m < C\log n$. They have shown in details that Partition sort, with $\gamma = 2$, makes $\Theta(n\log^2 n)$ comparisons and $\Theta(n\log^2 n)$ exchanges in the worst case, and $n\log n + \mathrm{O}(n)$ comparisons and $\Theta(n\log n)$ exchanges on the average. For other values of γ, the details of performance analysis are unclear. However, they indicated that Partition sort, with $\gamma = 128$, is in practice an efficient version.

Quicksort due to Hoare [14] is efficient on average, but goes easily quadratic time on reasonable inputs, such as "almost sorted". To avoid extreme slowdowns on plausible inputs, Bentley and McIlroy [1] presented a pseudo-median of nine strategy which is a pseudo-median Tukey's "ninther", the median of the medians of three samples, each of three elements. Their Quicksort is not only robust, but also among the fastest derivatives of Quicksort. The main drawback of this algorithm is that it has no adaptivity and goes possibly quadratic time on some inputs [19].

Both Splaysort [21] and McIlroy's mergesort [20] seems adaptive with respect to almost all accepted measures of presortedness. However, they are not practical. This is because they have such weaknesses: the data structures