ELSEVIER

Theoretical Computer Science 242 (2000) 29–40

Theoretical Computer Science

www.elsevier.com/locate/tcs

Presorting algorithms: An average-case point of view

Hsien-Kuei Hwang [a,*], Bo-Yin Yang [b], Yeong-Nan Yeh [c]

[a] *Institute of Statistical Science, Academia Sinica, Taipei 11529, Taiwan*
[b] *Department of Mathematics, Tamkang University, Taipei County, Tamsui 251, Taiwan*
[c] *Institute of Mathematics, Academia Sinica, Taipei 115, Taiwan*

Received December 1997; revised May 1998
Communicated by H. Prodinger

Abstract

We introduce the concept of presorting algorithms, quantifying and evaluating the performance of such algorithms with the average reduction in number of inversions. Stages of well-known algorithms such as Shellsort and quicksort are evaluated in such a framework and shown to cause a meaning drop in the inversion statistic. The expected value, variance and generating function for the decrease in number of inversions are computed. The possibility of "presorting" a sorting algorithm is also investigated under a similar framework.

Keywords: Presorting; Measure of presortedness (*mop*)

1. Introduction

Sorting is generally considered to be one of the most fundamental problems in computer science. It is omnipresent and inevitable in almost any serious application, and it is said that about a quarter of computer cycles used are spent sorting. Furthermore, it is a suitable "prototype problem" – a problem that is easily modeled and has good mathematical properties, as well as one which combinatorialists, statisticians, probabilists and computer scientists has long been studying, providing a steady source of good problems and theoretical innovations. See [7, 12, 13] for more information.

Since each of the major families of sorting algorithms has its strengths and weaknesses and none is "universally" efficient, the idea of "adaptive sorting algorithms" or "input sensitive algorithms" naturally arises. Several algorithms have been devised to take into account and exploit the existing order of the input sequence. Burge [1]

* Corresponding author.
E-mail address: hkhwang@stat.sinica.edu.tw (H. Hwang).

PII: S0304-3975(98)00181-9

first incorporated such an idea into the analysis of sorting algorithms; Mehlhorn [10] proposed a more concrete realization by giving a sorting algorithm "smoothly adaptive" to the number of inversions I_n of the input sequence (with n elements), defined as the number of disordered pairs. Roughly, if the input has few number of inversions, say, $I_n = O(n)$, then the algorithm is linear; the algorithm remains $O(n \log n)$ when the number of inversions increases up to $O(n^2)$. These ideas were then formalized and synthesized in Mannila's fine paper [9] on measures of presortedness (here and throughout, *mop*).

Following Mehlhorn and Mannila, a number of authors considered this adaptive aspect of sorting algorithms. The general pattern of approach is to first devise a new *mop*, to prove that it is not compatible with existing ones, and then to design an optimal sorting algorithm with respect to this *mop*. Parallel and randomized extensions of the problem have also been studied; see the survey by Petersson and Moffat [11] and the references therein.

In this paper, we take a different point of view by considering *presorting algorithms*. Instead of passively measuring the quantity of presortedness of the input, we actively "create" sortedness of the input by performing certain simple procedures with an aim to decrease the quantity of some *mop*. To give a rough analogy, a barber routinely spray the to-be-shorn hair with water. The idea is to make the hair easier to handle before the shears are applied. Note that "preprocessing" a problem is in fact a time-honored approach, and not at all a conflicting concept to measuring presortedness.

While preprocessing has been widely applied in diverse problems, the usefulness of its application to problems as fundamental (and simple) as sorting is not obvious. At an abstract level, we may regard the heap construction as a preprocessing unit for heapsort. Likewise, the collection of ascending runs in natural mergesort (cf. [7]) is also a preprocessing procedure. We do not, however, view the construction of the binary search tree as a preprocessing for sorting, the difference here being that the construction itself costs more than the steps that remain – in this case the in-order traversal – and is thus not efficient in the following sense.

In general, a preprocessing algorithm, especially for a simple problem as sorting, must be *simple* and *efficient*. Simplicity means easily programmable and efficiency (for sorting) signifies $O(n)$ operations on average. Obviously, an $O(n \log n)$ preprocessing for sorting does not make much sense.

While *mop* prefers nearly sorted data or certain skewed distributions of the input, our presorting algorithms prefer "random input". Our treatment of presorting algorithms is from an average-case point of view. We first fix a *mop* and then consider simple "devices" or operations capable of reducing the quantity of this *mop*. We thus study the average number of reductions in this *mop*.

Take a simple example: if we preform a comparison between the first and the last keys of a random input with n elements, followed by a possible exchange if they are out of order, the average reduction in number of inversions is given by $\frac{1}{3}n - \frac{1}{6}$ (assuming a uniform probability model on permutations of n elements). Repeating the same procedure once for the second and the next-to-last keys results in $\frac{2}{3}n - 1$ decrease

in inversions on average, etc. (see next section for details). Such simple operations are thus of much benefit to sorting.

From a more practical viewpoint, there is a different approach to these problems. If we fix a sorting algorithm, we may ask questions as how to tune it so that it is less input sensitive (this question received less attention) and, more relevant to this paper, how to preprocess inputs to increase the average efficiency of the algorithm. For example, how to preprocess the input so that it reduces the average number of comparisons used by quicksort? We will show that simple procedures such as one-step median-of-3 results in a decrease of $\frac{1}{6}n - \frac{1}{2}$ $(n \geqslant 3)$ on average in number of comparisons, with only $\frac{2}{3}$ more comparisons at the very first partitioning step!

Another widely used technique for recursive algorithms with a similar effect or character (*increasing the efficiency of an algorithm by simple procedures at the early stage*) is to tune (modify or even replace) the algorithm for small subfiles. We show that improvements of the standard top-down recursive mergesort at, say, $n = 5$ results in linear number of decrease for the avarege total cost with explicitly computable expression involving periodic functions.

2. Presorting algorithms for inversions

In this section, we develop the above ideas by way of several examples concentrating on the inversion statistic which is a prototype for *mop*'s. The number of inversions is an often-used measure of presortedness (hence, randomness). It is known in statistics as *Kendall's* τ and one of the most studied (by statisticians, combinatorialists, etc.) statistics.

As we stated before, the most obvious presorting algorithm consists of comparing a pair of keys, and switching them if they are out of order.

Let S_n be the symmetric group of order n. For $i > j$ we define $\mathsf{SW}_{i,j} : S_n \to S_n$ for $\sigma \in S_n$,

$$\mathsf{SW}_{i,j}(\sigma) := \begin{cases} \sigma, & \sigma_i < \sigma_j, \\ (i\ j)\,\sigma, & \sigma_i > \sigma_j. \end{cases}$$

In other words, we exchange $\sigma(i)$ and $\sigma(j)$ if out-of-order. When $i < j$, we define $\mathsf{SW}_{i,j} := \mathsf{SW}_{j,i}$.

Intuitively, a presorting algorithm should at least cause the average value of the *mop* under consideration to drop an amount commensurate with the effort. For example, for a constant-time action on a permutation in S_n, we expect a linear reduction for the expected value of I_n and for a linear-time action on a permutation in S_n, a quadratic reduction in number of inversions is expected.

We assume *throughout this paper that a uniform probability measure is assigned on the set* S_n. Thus, I_n is a random variable. It is known that the probability generating

function of I_n is given by (cf. [7])

$$I_n(z) := \mathrm{E}(z^{I_n}) = \frac{(1+z)(1+z+z^2)\cdots(1+z+\cdots z^{n-1})}{n!} \quad (n \geqslant 1),$$

from which it follows that

$$\mathrm{E}(I_n) = \frac{n(n-1)}{4}, \qquad \mathrm{Var}(I_n) = \frac{n(n-1)(2n+5)}{72} \quad (n \geqslant 2).$$

The distribution of I_n is asymptotically normal.

For $\sigma \in S_n$, let $P_k = \mathsf{SW}_{1,n}\mathsf{SW}_{2,n-1}\ldots\mathsf{SW}_{k,n-k+1} : S_n \to S_n$ and write $\Delta_k = \Delta_k(\sigma) = I_n(\sigma) - I_n(P_k(\sigma))$, where $1 \leqslant k \leqslant \lfloor n/2 \rfloor$.

Theorem. *We have*

$$\sum_{\sigma\in S_n} z^{I_n(P_k(\sigma))} = 2^k \left[\prod_{j=1}^{k} \left(1 + 2z + 3z^2 \cdots + (n-2j+1)z^{n-2j}\right)\right] \left[\prod_{j=1}^{n-2k} \frac{1-z^j}{1-z}\right], \quad (1)$$

$$\sum_{\sigma\in S_n} z^{\Delta_k(\sigma)} = (n-2k)! \prod_{j=0}^{k-1} \left[\binom{n-2j}{2} + \sum_{\ell=1}^{n-2j-1} (n-2j-\ell)z^{2\ell-1}\right]. \quad (2)$$

From these generating functions, we easily derive results for the first two moments of Δ_k.

Corollary. *The mean and variance of the decrease in the I_n statistic for the comparison-and-switching presorting algorithms P_k are given by*

$$\mathrm{E}(\Delta_k) = \frac{kn}{3} - \frac{k(2k-1)}{6},$$

$$\mathrm{Var}(\Delta_k) = \frac{2}{9}kn^2 - \frac{2}{9}kn(2k-1) + \frac{k}{108}(32k^2 - 24k - 29).$$

In particular, we have, for a single switch operation,

$$\mathrm{E}(\Delta_1) = \frac{n}{3} - \frac{1}{6}, \qquad \mathrm{Var}(\Delta_1) = \frac{2}{9}n^2 - \frac{2}{9}n - \frac{7}{36};$$

for two switch operations,

$$\mathrm{E}(\Delta_2) = \frac{2n}{3} - 1, \qquad \mathrm{Var}(\Delta_2) = \frac{4}{9}n^2 - \frac{4}{3}n - \frac{17}{18};$$

and for $\lfloor n/2 \rfloor$ operations,

$$\mathrm{E}(\Delta_{\lfloor n/2 \rfloor}) = \begin{cases} \dfrac{n(n+1)}{12}, & \text{if } n \text{ is even,} \\ \dfrac{(n-1)(n+2)}{12} & \text{if } n \text{ is odd,} \end{cases} \quad (3)$$

$$\mathrm{Var}(\Delta_{\lfloor n/2 \rfloor}) = \begin{cases} \dfrac{n(8n^2+12n-29)}{216} & \text{if } n \text{ is even,} \\ \dfrac{(n-1)(8n^2+20n-9)}{216} & \text{if } n \text{ is odd.} \end{cases}$$

We start with a simple lemma.

Lemma. *The single operation* $\mathsf{SW}_{i,j}$ *reduces the number of inversions by* $\frac{1}{3}(j-i)+\frac{1}{6}$ *on average.*

Proof. Take $\sigma \in S_n$. Half the time (when $\sigma_i < \sigma_j$ and $i < j$) $\mathsf{SW}_{i,j}(\sigma) = \sigma$; the other half of the time the number of inversions is reduced by

$$1 + \#\{k | i < k < j, \sigma_i > \sigma_k > \sigma_j\}.$$

To find the expected value of the last expression we need only observe that for each k between i and j the conditional probability that $\sigma_i > \sigma_k > \sigma_j$ (given that the switch occurs) is exactly $\frac{1}{3}$. □

The lemma says that if we limit our presorting actions to a sequence of comparisons between two elements (and switching when they are out of order), then the best presorting action on S_n with one single comparison (and switch) with respect to I_n is $\mathsf{SW}_{1,n}$, $\mathrm{E}(\Delta_1) = (2n-1)/6$. Similarly, we can show that the best two-comparison presorting with respect to I_n is either $\mathsf{SW}_{1,n}\,\mathsf{SW}_{2,n-1}$ or $\mathsf{SW}_{1,n-1}\,\mathsf{SW}_{2,n}$, either of which leads to $\mathrm{E}(\Delta_2) = \frac{2}{3}n - 1$ and that the maximum possible reduction in $\mathrm{E}(I_n)$ when we are limited to under $\lfloor n/2 \rfloor$ comparisons is given by (3). Note that, intuitively, it is not efficient if the position of any element in a sequence is moved more than once.

Proof of the Theorem. We only prove the theorem for $k = 1$, the idea extending easily to any $k \leqslant \lfloor n/2 \rfloor$ by applying the same arguments to the "inner cycles".

For the first part ((1) with $k = 1$), the factor of two is just multiplicity of the mapping. Clearly, if $\pi = \mathsf{SW}_{1,n}(\sigma)$ then we have $\pi_1 < \pi_n$, any other π_j that falls within the range $\pi_1 < \pi_j < \pi_n$ contributes *no inversions* to the total, and any outside *one inversion* each. Hence, if π_1 and π_n are adjacent ($n-1$ possibilities) there are $(n-2)$ inversions caused by π_1 and π_n; similarly, the pair contributes $(n-3)$ inversions if spaced two part, etc., down to no inversions at all if they are 1 and n, respectively. Inversions caused by the other π_j's are independent of the above and can be read off from the formula.

For the second part ((2) with $k = 1$), if $\sigma_1 < \sigma_n$ ($\binom{n}{2}$ cases) then I_n does not change at all! Otherwise, if these two numbers are adjacent then we only save one inversion ($n-1$ cases), and for each j that satisfies $\sigma_1 > \sigma_j > \sigma_n$ we save two extra inversions, so $n-2$ cases of saving 3 inversions, $n-3$ cases for 5, etc., up to exactly 1 case for saving $4n-3$ inversions, namely when $\sigma_1 = n$, $\sigma_n = 1$. □

3. Sorting as presorting

From the preceding discussions, it is obvious that the operation $P_{\lfloor n/2 \rfloor}$ has the same effect as a Shellsort with increment $\lfloor n/2 \rfloor$ as far as the average number of inversions

is concerned. And the results reveals essentially that initial large increment of shellsort is by its very own nature an efficient presorting method, which accounts for a part of its good performance in the medium-sized sample range.

We can apply this same idea to other sorting algorithms. In this section, we consider the effect of one-pass quicksort in terms of the number of inversions.

The average number of inversions of a random permutation generated by one-pass median-of-$(2t+1)$ quicksort partitioning is given by [1]

$$\begin{aligned}
&\frac{1/2}{\binom{n}{2t+1}}\left(\sum_{1\leqslant m\leqslant n}(m-1)(m-2)\binom{m-1}{t}\binom{n-m}{t}\right)\\
&\quad=\frac{1/2}{\binom{n}{2t+1}}[z^n]\frac{z^{2t-2}(2z^2+4tz+t^2-t)}{(1-z)^{2t+4}}\\
&\quad=\frac{t+2}{4(2t+3)}n^2-\frac{5t+6}{4(2t+3)}n+\mathrm{O}_t(1),
\end{aligned}$$

for $t\geqslant 0$, where the implied constants in the O-term depends on t.

Thus, for one round of quicksort using median-of-$(2t+1)$ on a random permutation in S_n, the average reduction in I_n is given by

$$\frac{t+1}{4(2t+3)}n^2-\frac{3(t+1)}{4(2t+3)}n+\mathrm{O}_t(1)\quad(n\to\infty),$$

for each integer $t\geqslant 0$.

Note that the leading constant of the quadratic term tends to $\frac{1}{8}$ as t becomes large, implying that about half the number of inversions can be saved by taking slightly larger t.

4. Presorting algorithms for quicksort

There has been studies (cf. [8]) regarding the use of median-of-$(2t+1)$-quicksort with variant $t\geqslant 0$ for different stages which may also depend on n the size of the problem. We consider instead here the case of a single round of partition when t is fixed and small.

Let $H_n=\sum_{1\leqslant j\leqslant n}1/j$ denote the harmonic numbers. If we use the formula for the cost (average number of comparisons) of quicksorting a random input of n elements

$$q_n=2(n+1)H_n-4n\qquad(n\geqslant 1),$$

and use custom algorithms for minimum-average-comparison selections (cf. [7]) with $2\frac{2}{3}$ comparisons for median of three, $5\frac{13}{15}$ for median of five, and $9\frac{32}{105}$ for median of 7, and choosing the sample from both the beginning and the end to minimize further

[1] The notation $[z^n]f(z)$ represents the coefficient of z^n in the Taylor expansion of f.

comparisons and data movement, then we have

$$q_n^{(3)} = n - \frac{1}{3} + \frac{12}{n(n-1)(n-2)} \sum_{j=1}^{n-1} j(n-1-j)q_j.$$

$$= 2(n+1)H_n - \frac{25}{6}n + \frac{1}{2} \;=\; q_n - \frac{n}{6} + \frac{1}{2} \quad (n \geqslant 3).$$

Thus, on average, we do $\frac{1}{6}n - \frac{1}{2}$ $(n \geqslant 3)$ fewer comparisons just by using $\frac{2}{3}$ more comparisons at the very first partitioning stage!

Similarly, we have

$$q_n^{(5)} = 2(n+1)H_n - \frac{127}{30}n + \frac{49}{30} \quad (n \geqslant 5),$$

$$q_n^{(7)} = 2(n+1)H_n - \frac{1793}{420}n + \frac{85}{28} \quad (n \geqslant 7).$$

In other words, we save on average $\frac{7}{30}n - \frac{49}{30}$ (resp. $\frac{113}{420}n - \frac{85}{28}$) number of comparisons by using $1\frac{13}{15}$ (resp. $3\frac{32}{105}$) more comparisons at the first partitioning stage. Note whereas $q_n^{(3)} \leqslant q_n$ for all n that makes the summation meaningful, we need $n \geqslant 7$ in order to have $q_n^{(5)} \leqslant q_n$, and it is only for $n \geqslant 12$ that we have $q_n^{(7)} \leqslant q_n$.

We can of course continue this process. Take for example the possibility of taking two rounds of median-of-three quicksort, which can be derived from the $q_{n,1}^{(3)} := q_n^{(3)}$ above:

$$q_{n,2}^{(3)} := n - \frac{1}{3} + \frac{12}{n(n-1)(n-2)} \left(\sum_{3 \leqslant j < n} j(n-1-j)q_{j,1}^{(3)} + 2(n-3) \right).$$

$$= q_n - \frac{n-5}{3} - \frac{4(2n-5)}{n(n-1)(n-2)}.$$

In fact continuing in this manner, with

$$q_{n,k}^{(3)} := n - \frac{1}{3} + \frac{12}{n(n-1)(n-2)} \left(\sum_{3 \leqslant j < n} j(n-1-j)q_{j,k-1}^{(3)} + 2(n-3) \right),$$

and using the identities (for $n \geqslant 3$)

$$\frac{12}{n(n-1)(n-2)} \sum_{3 \leqslant j < n} j(j+1)(n-1-j) = \frac{n+1}{6} - \frac{4(4n-11)}{n(n-1)(n-2)},$$

$$\frac{12}{n(n-1)(n-2)} \sum_{3 \leqslant j < n} j(n-1-j) = 2 - \frac{12(3n-8)}{n(n-1)(n-2)},$$

we can write

$$q_{n,k}^{(3)} = q_n - \frac{k}{6}(n+1) + c(k) - E_{n,k},$$

where $c(0) = 0$,

$$c(k) = 2c(k-1) + \frac{2}{3} = \frac{2}{3}(2^k - 1) \qquad (k \geqslant 1),$$

and $E_{n,1} = 0$,

$$E_{n,k} = \frac{12}{n(n-1)(n-2)} \left(\sum_{3 \leqslant j < n} j(n-1-j)E_{j,k-1} + \frac{3 \cdot 2^k - 4k - 2}{3} n - \frac{2^{k+3} - 11k - 5}{3} \right),$$

for $k \geqslant 2$.

From the estimate

$$E_{n,2} = \frac{4(2n-5)}{n(n-1)(n-2)} \sim 8n^{-2},$$

we deduce that

$$E_{n,3} \sim 8\frac{12 \log n}{n^2},$$

and by induction

$$E_{n,k} \sim 8\frac{(12 \log n)^{k-2}}{(k-2)!n^2},$$

for each $k \geqslant 2$.

Thus, the correction term $E_{n,k}$ is uniformly small for $k = \mathrm{O}(1)$, as $n \to \infty$ and

$$q_{n,k}^{(3)} = q_n - \frac{k}{6}(n+1) + \frac{2}{3}(2^k - 1) + \mathrm{O}(n^{-2+\varepsilon}),$$

for $k = \mathrm{O}(1)$.

In the same vein, we have

$$q_{n,k}^{(5)} = q_n - \frac{7k}{30}(n+1) + \frac{56(2^k - 1)}{30} + \mathrm{O}(n^{-2+\varepsilon});$$

$$q_{n,k}^{(7)} = q_n - \frac{113k}{420}(n+1) + \frac{1388(2^k - 1)}{420} + \mathrm{O}(n^{-2+\varepsilon}),$$

for any $\varepsilon > 0$ and $k = \mathrm{O}(1)$.

As in the presorting algorithms for inversions discussed earlier, *just one* early stage median-of-$(2t+1)$ for quicksort results in a fairly large saving in cost with later such stages gaining less and less. *This is intuitively in accordance with the law of diminishing returns.*

5. Improvement of recursive algorithms on small subfiles

When a recursive algorithm is not efficient on small subfiles, it is customary to use more straightforward alternatives in order to improve the efficiency of the algorithm.

A well-known example is to resort to insertion sort when quicksorting subfiles of size smaller then, say, 9 (cf. [7, 12]). In this section, we show that such a simple "presorting" idea is also useful for the top-down mergesort (cf.[3]).

Roughly, to sort an input of n elements, divide it into two subfiles of sizes $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$, respectively, sort these two recursively, and then use, say, the linear merge algorithm to merge the two sorted files. To "presort" mergesort we stop the recursive call for subfiles of size less than $N \geqslant 2$ and use more efficient sorting algorithms for small files. The average number of comparisons used to mergesort a random permutation (assuming each of $n!$ permutations of n elements is equally likely) of n elements is given by the recurrence (cf. [3])

$$f_n = f_{\lfloor n/2 \rfloor} + f_{\lceil n/2 \rceil} + e_n \quad (n > N), \tag{4}$$

with suitable initial values of f_n, $1 \leqslant n \leqslant N$, and

$$e_n = n - \frac{\lfloor n/2 \rfloor}{\lceil n/2 \rceil + 1} - \frac{\lceil n/2 \rceil}{\lfloor n/2 \rfloor + 1} \qquad (n \geqslant 1).$$

If we take $N = 1$, direct computations show that f_n is optimal (in terms of the expected number of comparisons) only for $n \leqslant 4$ (cf. [7, p. 195]). What is the expected number of comparisons reduced if we use more efficient sorting algorithms for $5 \leqslant n \leqslant N$? We show that a linear number will generally be dropped even in the case when we reduce only the number of comparisons for sorting five elements ($N = 5$).

To solve (4) for general N, we solve (4) for $N = 2$ by inserting the differences in value in the e_n's. Thus, we replace e_n in (4) by $e'_n = e_n - \delta_n$, where $\delta_n > 0$ for $n = 5, \ldots, N$. By the analytic approach of Flajolet and Golin [3], we have the integral representation

$$f_n = \frac{1}{2\pi \mathrm{i}} \int_{2-\mathrm{i}\infty}^{2+\mathrm{i}\infty} \frac{n^{s+1}}{s(s+1)} \frac{\Xi(s)}{1-2^{-s}} \, \mathrm{d}s,$$

where (defining $e_0 = 0$ and $\Delta\nabla e_n = e_{n+1} - 2e_n + e_{n-1}$)

$$\Xi(s) = \sum_{n \geqslant 1} \frac{\Delta\nabla e_n}{n^s} + D_N(s) \quad (\Re s > -1),$$

with ($\delta_j = 0$ for $j < 5$ and $j > N$)

$$D_N(s) = \sum_{4 \leqslant j \leqslant N+1} \frac{\Delta\nabla \delta_j}{j^s}.$$

Thus, in particular,

$$D_5(s) = -\frac{\delta_5}{4^s} + \frac{2\delta_5}{5^s} - \frac{\delta_5}{6^s},$$

$$D_6(s) = -\frac{\delta_5}{4^s} + \frac{2\delta_5 - \delta_6}{5^s} + \frac{2\delta_6 - \delta_5}{6^s} - \frac{\delta_6}{7^s}.$$

Note that $D_N(0) = 0$.

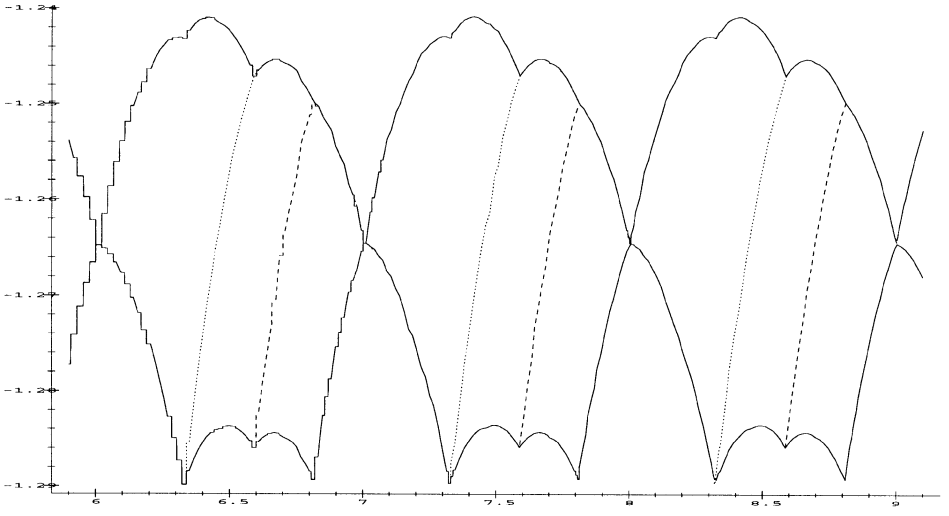

Fig. 1. Fluctuation terms of mergesort with and without "preprocessing". (Plotted are the coefficients of the linear term.)

Applying the techniques used in Hwang [6], we deduce that the average number of comparisons dropped is given exactly by

$$-n\left(\frac{\Delta'(0)}{\log 2}+\sum_{4\leqslant j\leqslant N+1}\Delta\nabla\delta_j\cdot A(\log_2(n/j))\right), \tag{5}$$

where

$$A(u)=1-\{u\}-2^{1-\{u\}}.$$

For example, if we use the merge insertion sort of Ford and Johnson (cf. [7, p. 186]) for $n=5$ (and $N=5$), then we have $\delta_5=7/30$ and economize

$$n\left(\frac{25}{24}+A(\log_2(n/4))-2A(\log_2(n/5))+A(\log_2(n/6))\right)$$

number of comparisons. If we use the same sorting algorithm for both $n=5$ and $n=6$ (and $N=6$), then the expected number of comparisons reduced is given by (5) with $\delta_5=\delta_6=7/30$. Similarly, using the Ford–Johnson algorithm for $n=5,6,7$ results in δ_5 and δ_6 as given above and $\delta_7=29/105$. A graphical rendering of these examples is given in Fig. 1.

6. Remarks

The preceding discussions are not restricted to just inversions as the sole *mop*, nor to quicksort or mergesort as the sole sorting algorithm to work with. However, they are certainly the most instructive examples. We observe that *mop*'s such as the number

of runs with ranges of order n is not very useful when handling presorting methods (however useful they may be for other purposes). In general, as in nonparametric inference, a right-invariant metric with small variance, say, linear or less than linear, is unsuitable for general use (cf. [2, 4]). On the other hand, computations of expectation values in drop of $DS := \sum_{1\leqslant j\leqslant n} |j-\pi(j)|$ or $\sum_{1\leqslant j\leqslant n-1} |\pi(j)-\pi(j+1)|$ can be made in a similar manner as for I_n (but not for generating functions which are intrinsically much harder). For example, applying $\mathsf{sw}_{1,n}$ to a permutation in S_n decreases the expected value of DS by

$$\mathrm{E}(\Delta(DS)) = \tfrac{2}{3}n + \mathrm{O}(1).$$

For,

$$\Delta(DS) := DS(\sigma) - DS(\mathsf{sw}_{1,n}(\sigma)) = 2[\max(\sigma_1, \sigma_n) - \min(\sigma_1, \sigma_n)],$$

and we know that max of two random elements in $[1, n]$ has an expected value around $2n/3$, the min about $n/3$.

When several presorting algorithms are available, one may further *quantify the goodness* (*or efficiency*) of these algorithms by suitable definitions. For example, for our compare-and-switch algorithms, we may define the *efficiency factor* $\alpha_{n,k}$ of P_k on an input of n elements as follows:

$$\alpha_{n,k} := \frac{3\mathrm{E}(\Delta_k)}{n \cdot (\text{cost of } P_k)}.$$

Thus

$$\alpha_{n,k} = 1 - \frac{2k-1}{2n} \to 1,$$

when $k = \mathrm{o}(n)$; and

$$\alpha_{n,k} \to 1 - \alpha,$$

when $k = \alpha n$, $0 < \alpha \leqslant \frac{1}{2}$. Since $\alpha_{n,k} > \alpha_{n,k+1}$, we may say that P_k is more efficient than P_{k+1}.

Acknowledgements

This research was partially supported by the National Science Council under Grants NSC-86-2115-M-001-004, and NSC-86-2115-M-032-010.

References

[1] W.H. Burge, Sorting, trees, and measures of order, Inform. and Control 1 (1958) 181–197.
[2] P. Diaconis, R.L. Graham, Spearman's footrule as a measure of disarray, J. Roy. Soc. Statist. Ser. B 39 (1977) 262–268.

[3] P. Flajolet, M. Golin, Mellin transforms and asymptotics: the mergesort recurrence, Acta Inform. 31 (1994) 673–696.
[4] M.A. Fligner, J.S. Verducci (Eds.), Probability models and statistical analysis, for ranking data, Lecture Notes in Statistics, vol. 80, Springer, Berlin, 1993.
[5] D.H. Greene, D.E. Knuth, Mathematics for the Analysis of Algorithms, Birkhäuser, Boston, 1981.
[6] H.-K. Hwang, Asymptotic expansions of mergesort recurrences, Acta Inform., to appear.
[7] D.E. Knuth, The Art of Computer Programming, vol. III, Sorting and Searching. Addison-Wesley, Reading, MA. 1973.
[8] C.C. McGeoch, J.D. Tygar, Optimal sampling strategies for quicksort, Random Struct. Algorithms 7 (1995) 287–300.
[9] H. Mannila, Measures of presortedness and optimal sorting algorithms, IEEE Trans. Comput. 34 (1985) 318–325.
[10] K. Mehlhorn, Sorting presorted files, in: Theoretical Computer Science (4th GI Conference, Aachen, 1979), pp. 199–212, Lecture Notes in Computer Science, vol. 67, Springer Berlin, 1979.
[11] O. Petersson, A. Moffat, A framework for adaptive sorting, Discrete Appl. Math. 59 (1995) 153–179.
[12] R. Sedgewick, Algorithms, 2nd ed., Addison-Wesley, Reading, MA, 1988.
[13] J.S. Vitter, P. Flajolet, Average-case analysis of algorithms and data structures, in: Handbook of Theoretical Computer Science, vol. A, Algorithms and Complexity, Elsevier, Amsterdam, 1990, pp. 431–524.