IMPLEMENTATION OF SAMPLESORT:
A MINIMAL STORAGE TREE SORT

J. G. PETERS and P. S. KRITZINGER

Abstract.

An implementation of samplesort, a generalization of minimal-storage tree-sorting, is presented as an algorithm which is relatively insensitive to possible non-random permutations of the elements to be sorted. A brief analytical discussion and the results of extensive empirical tests are included. It is claimed that with the particular choice of samplesize, samplesort is probably one of the most efficient, if not the most efficient sorting algorithm known.

Keywords: internal sorting, quicksort, samplesort, minimal-storage sorting, tree-sort.

1. Introduction.

The algorithm described here is an implementation of the procedure suggested by W. D. Frazer and A. C. McKellar [2], which in turn is a generalization of minimal storage Quicksort first proposed by C. A. R. Hoare [4].

The reader is referred to the paper by Frazer and McKellar for a detailed description of the procedure. We will only briefly discuss the algorithm and indicate where our implementation differs from their suggestions.

2. Description.

The algorithm proceeds in three phases. In the first phase a sample of size $s$ elements is chosen from the $n$ elements of the input sequence, and subsequently sorted. The sample should be large enough to closely approximate the cumulative distribution function of the input sequence. Remarks concerning the choice of $s$ are deferred until later. Ideally the sample-elements should be chosen at random from the input sequence, but elements spaced at constant intervals $[n/s]$ are chosen here to facilitate implementation.

In the second phase the sorted sample is used to partition the input sequence resulting in a set of subsequences bounded by the elements of the sample. These subsequences may be considered as the leaves of a binary sort tree with the sample elements at the vertices. In fact the motivation for using samplesort is the characteristic that the sort tree so obtained will be more balanced than that obtained using Quicksort, in the sense that all the subsequences will have approximately the same number of elements.

Frazer and McKellar suggested two ways of implementing this second phase. In the first the sample elements are removed to an auxiliary array and, after the partitioning operation, merged with the remainder sequence. In the second technique the sample elements remain in the original sequence. A different method is used here which avoids the merge of the first method and the extra item transfers which is characteristic of the second.

In the version presented here a copy of the sample elements are moved to a temporary array. Partitioning is then carried out in a manner similar to that suggested by Frazer and McKellar in the first method. Except that now, after a sample element has been used to form a partition, its location in the temporary array is used to store a pointer to the endpoint of the subsequence just formed.

In the third phase of the algorithm the individual subsequences are sorted using a (not necessarily) different algorithm.


Using minimum expected number of comparisons as the criterion of optimality, very little theoretical argument can be provided for either the particular form or the best choice of the samplesize $s$. Frazer and McKellar discuss the matter at some length. We derived our choice using the following simple approach:

Let $C$ be the number of comparisons required to sort $n$ items using samplesort. Then the expected value of $C$ is given by

$$E(C) = E(C_1) + E(C_2) + E(C_3)$$

where $C_1$ is the number of comparisons required to sort the sample, $C_2$ that number required to insert the sample and $C_3$ is the number of comparisons required to sort all the subsequences. M. van Emden [8] proved that

$$E(C_1) = 1.14s \log_3 s$$