CS81012-R

USORT: AN EFFICIENT HYBRID OF DISTRIBUTIVE PARTITIONING SORTING

D.C.S. Allison
M.T. Noga
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

ABSTRACT

A new hybrid of Distributive Partitioning Sorting is described and tested against Quicksort on uniformly distributed items. Pointer sort versions of both algorithms are also tested.

Key Words: Sorting, distributive partitioning, quicksort.

1. INTRODUCTION

Distributive Partitioning Sorting (DPS) has received a good deal of attention in the literature lately. The evidence so far indicates that DPS is in many cases the fastest known sorting technique. For example, tests by Kowalik and Yoo [1] show this technique to be roughly 70% faster than two efficient versions of Quicksort [2], [3] on an Amdahl V/6 computer for uniform and normally distributed inputs. One drawback of DPS is the space required to sort. In most implementations 3n-4n storage locations are needed for input strings of size n. Fortunately, most modern "mainframes" have several megabytes of storage, thus making DPS practical for inputs of length 100,000 or more.

Presently, there exist at least four different distributive sorting algorithms. The original method [4] is recursive and relies on median calculation for sorting. The remaining three methods [5], [6], [1] avoid expensive median calculation and thus are somewhat faster than the original method. Unfortunately, these hybrids have all been tested on different hardware. Thus, as of yet, no empirical evidence exists which would be of help in determining the differences in speed amongst the three methods.

The purpose of this study is to present another computationally efficient hybrid of the original method. We have coded this new method in FORTRAN and tested its performance against Sedgewick's Quicksort [2]. In addition, pointer sort versions of the new method and Sedgewick's Quicksort have also been implemented and compared.

2. THE METHOD

The sort, which we label Usort, because it performs most efficiently on uniform distributions of items, requires three passes, first a distributive pass and then two comparison based passes. In the first pass the n items are partially sorted into $\lfloor n/m \rfloor$[1] boxes, an item A_i being placed into box j according to the formula

$$j := \lfloor (A_i - \min)/(\max - \min) * (\lfloor n/m \rfloor - e) + 1 \rfloor. \quad (1)$$

The quantity e (taken to be very small relative to the size of n) is needed to insure that the maximum item is placed into box $\lfloor n/m \rfloor$ instead of box $(\lfloor n/m \rfloor + 1)$ [7]. The contents of two consecutive boxes are such that all the items in the first one are guaranteed to be smaller than all the times in the second one.

For the second pass each box which contains k or more items is partitioned by Sedgewick's "median-of-three" Quicksort until all partitions are of size k-1 or smaller. The value chosen for k should be about 9, although any value between 6 and 20 may best optimize performance on any particular machine. The third pass consists of a single Insertion sort (see [8] for a description of Insertion sorting) over the entire vector. In this way the stacking overhead associated with an Insertion sort on each box or partition can be avoided. Sedgewick [2] has already used this technique with good success in conjunction with Quicksort.

[1]$\lfloor x \rfloor$ is the greatest integer smaller $\overset{\text{than}}{\wedge}$ or equal to x.

3. IMPLEMENTATION AND STORAGE REQUIREMENTS

The elements are not moved during the first pass. Instead, a singly linked-list is used to represent the items of each box. The linked-list requires a total of $\lfloor n/m \rfloor + n$ storage locations: $\lfloor n/m \rfloor$ for the list heads and n locations for the links. Assuming that the items are in array A, efficient psuedo-code for distributing the items into $\lfloor n/2 \rfloor$ boxes would be

```
ndiv2 := n/2;
FOR i:= 1 TO ndiv2 DO     (*initialize list heads*)
   list_head [i]:=0;
constant:= (ndiv2 - .001)/(A[max] - A[min]);    (*e has the value .001*)
FOR i:= 1 TO n DO
    BEGIN
      j:= (A[i] - A[min]) * constant + 1.0;    (*the result is truncated*)
      link [i]:= list_head [j];
      list_head [j]:= i
    END
```

As a result of this computation each empty list will have a list head equal to zero and each non-empty list will have a terminating zero-link. By making one pass through all the lists, the contents of the boxes may be quickly rearranged into a destination array B where passes two and three may be efficiently carried out. The total array storage required is $(3n + \lfloor n/m \rfloor)$.

4. WORST-CASE AND AVERAGE-CASE TIME COMPLEXITY

The worst-case complexity will occur in the unlikely event that the value of the items follow a factorial distribution. All of the items with the exception of the largest will fall into the first box and Quicksort will have to be applied to (n-1) elements. Since the worst-case time complexity of Quicksort is $O(n^2)$ for groups of size n [8], it follows that the worst-case complexity of Usort is also $O(n^2)$.

We analyze the expected-case time complexity under the assumption that the input sequence consists of uniformly distributed numbers. It takes $O(n)$ time to find A(min), A(max), initialize the list heads, distribute the items into the created intervals, and rearrange the items into a form suitable for the second and third passes of the algorithm. For the second pass the time to sort a single box consisting of i items will be proportional to $i\log_2 i$, the expected-case of Quicksort [8]. Since the input is uniformly distributed the probability that an item belongs to a given group is $1/(n/m)=m/n$. The probability that a single box will consist of i items is a binomial distribution

$$P(i)=\binom{n}{i}\left(\frac{m}{n}\right)^{i}\left(1-\frac{m}{n}\right)^{n-i}$$

The expected time to sort a single box of k or more items is

$$\sum_{i=k}^{n} i\log_2 i\binom{n}{i}\left(\frac{m}{n}\right)^{i}\left(1-\frac{m}{n}\right)^{n-i}.$$

The time to sort all the boxes is, therefore,

$$n/m \sum_{k}^{n} i\log_2 i \binom{n}{i}\left(\frac{m}{n}\right)^{i}\left(1-\frac{m}{n}\right)^{n-i}$$

$$= n/m\left[k\log_2 k\binom{n}{k}\left(\frac{m}{n}\right)^{k}\left(1-\frac{m}{n}\right)^{n-k} + (k+1)\log_2(k+1)\binom{n}{k+1}\left(\frac{m}{n}\right)^{k+1}\left(1-\frac{m}{n}\right)^{n-k-1} + \ldots\right]$$

$$= n/m\left[k\log_2 k\,\frac{n(n-1)\ldots(n-k+1)}{k!}\,\frac{m^k}{n^k}\left(1-\frac{m}{n}\right)^{n-k} + (k+1)\log_2(k+1)\,\frac{n(n-1)\ldots(n-k+2)}{(k+1)!}\right.$$
$$\left.\frac{m^{k+1}}{n^{k+1}}\left(1-\frac{m}{n}\right)^{n-k-1}+\ldots\right]$$

$$= n/m\left[\frac{(k\log_2 k)m^k}{k!}\,\frac{n(n-1)\ldots(n-k+1)}{n^k}\left(1-\frac{m}{n}\right)^{n-k} + \frac{(k+1)\log_2(k+1)m^{k+1}}{(k+1)!}\,\frac{n(n-1)\ldots(n-k+2)}{n^{k+1}}\right.$$
$$\left.\left(1-\frac{m}{n}\right)^{n-k-1}+\ldots\right]$$

$$< n/m\left[\frac{(k\log_2 k)m^k}{k!}(1) + \frac{(k+1)\log_2(k+1)m^{k+1}}{(k+1)!}(1) + \ldots\right]$$

$$= n/m\left[\frac{k\log_2 k}{\frac{k}{m}\cdot\frac{k-1}{m}\cdots\frac{1}{m}} + \frac{(k+1)\log_2(k+1)}{\frac{k+1}{m}\cdot\frac{k}{m}\cdots\frac{1}{m}} + \ldots\right]$$

$$= n\left[\frac{\log_2 k}{\frac{k-1}{m}\cdots\frac{1}{m}} + \frac{\log_2(k+1)}{\frac{k}{m}\cdots\frac{1}{m}} + \ldots\right]. \qquad (1)$$

The sum inside the brackets converges for small m. Thus equation (1) is $O(n)$. After the second pass all partitions and boxes will have k-1 or less items. For Insertion sort the worst-case will be when there are $n/k-1$ of these groups, each of size k-1. The time to order all these groups is

$$\left(\frac{n}{k-1}\right)(k-1)^2 = n(k-1) = O(n),$$

since k will be small and fixed for any implementation of Usort. Summing over all steps the total time taken is $O(n)$.

In practical situations distributions which are uniform do not occur very frequently. However, many "real" applications involve the ordering of data which exhibit near uniform behavior. With regard to Usort, this means that after the first pass $O(n)$ boxes will contain at least one item, but with low probability no single box will overload (become too populous). For distributions meeting these requirements Usort will be very fast. In the event that several boxes contain a significant number of items (with respect to the size of the input vector), the time to sort will be reasonable because of Quicksort's $O(n\log_2 n)$ expected-case time complexity. Pass two is a "fail-safe" mechanism which insures against all but the most pathological cases.

5. MODIFICATIONS FOR POINTER SORTING

Usort is easily modified for pointer sorting [3]. After the first pass an array of length n is needed to initialize the pointers. The auxiliary array B referred to in section 3 is not needed. The storage requirements are therefore identical to the "straight-exchange of keys" version. Of course, in passes two and three, comparisons will take the form

$$A[pointer[i]] < A[pointer[j]]$$

with the pointers being exchanged depending upon the outcome of the test. This indirect reference causes more overhead because of the extra data movement; however, the expected-case time complexity of Usort with pointers is still O(n).

6. RESULTS AND DISCUSSION

The two algorithms described above, Usort (USORT) and Usort with pointers (UPSORT) were coded in FORTRAN and run on an IBM 3Ø32 (FORTX,OPT=2). These were tested against FORTRAN implementations of Sedgewick's Quicksort (QSORT) and Quicksort with pointers (QPSORT) [2]. Psuedo-random variates were generated over the interval (Ø,1) by IMSL uniform random number generator GGUBS [9]. 5 realizations of 1ØØ runs were made on all sample sizes. The timings taken were averaged and are summarized in Table I. Variances were calculated for each set of 5 test runs and appear in the parentheses at the right.

We experimented with different m and k before choosing values 2 and 8. As mentioned previously these values are not critical. For example with m=3 and k=10 the total time was only about 2% higher. However, we did find that Usort ran much slower when m=1. One reason for this behavior is that many of the lists were empty and maintenance of empty lists incurs extra overhead.

n	QSORT	USORT	QPSORT	UPSORT
250	39.4 (.24)	28.6 (.24)	58.8 (.16)	42.8 (.16)
500	87.4 (.64)	57.8 (.16)	130.2 (.16)	86.6 (.24)
1000	191.0 (3.20)	114.8 (.96)	286.2 (1.36)	175.4 (3.44)
2000	416.0 (.40)	233.4 (.64)	629.2 (18.16)	350.8 (3.76)
4000	894.0 (16.40)	472.0 (20.80)	1366.6 (184.24)	702.4 (46.64)

Table I. Sorting time (average of 5 realizations of 100 runs in hundredths of a second).

The results indicate that Usort is a very fast sorting method; 50-90% faster than Quicksort on the sample sizes used in the test. We would expect the performance to be even better on machines with fast floating point hardware such as the CDC Cyber Series [5]. Our results compare favorably with Kowalik and Yoo's. However, they used sample sizes of 5000, 10000, and 50000 items [1]. Besides speed, the advantages of Usort include its ease of implementation, avoidance of median calculation, and its three pass hierarchy.

Devroye and Klincsek [10] have shown that other distributions which have exponentially dominating tails can also be sorted in O(n) time by distributive partitioning. Among these are the normal, experimental gamma, beta, chi-square, and rectangular densities, plus all bounded densities with compact support. Some authors have downplayed the significance of distributive sorting methods [11] [12]. However, our results indicate to the contrary. We believe that any significant increase in the sorting of internal files (say on the order of 5% or more) is of major importance to the computing community.

REFERENCES

[1] Kowalik, J.S. and Y.B. Yoo. "Implementing a distributive sort program." Journal of Information and Optimization Sciences 2, no. 1, 1981, pp. 28-33.

[2] Sedgewick, R. "Implementing quicksort programs." CACM 21, no. 10, 1978, pp. 847-856.

[3] Loeser, R. "Some performance tests of 'Quicksort' and descendents." CACM 17, no. 3, 1974, pp. 143-152.

[4] Dobosiewicz, W. "Sorting by distributive partitioning." Info. Proc. Lett. 8, no. 4, 1979, pp. 168-169.

[5] Van der Nat, M. "A fast sorting algorithm, a hybrid of distributive and merge sorting." Info. Proc. Lett. 10, no. 3, 1980, pp. 163-167.

[6] Meijer, H. and S.G. Akl. "The design an analysis of a new hybrid sorting algorithm." Info. Proc. Lett. 10, no. 4-5, 1980, pp. 213-218.

[7] Allison, D.C.S. and M.T. Noga. "Selection by distributive partitioning." Info. Proc. Lett. 11, no. 1, 1981, pp 7-8.

[8] Wirth, N. Algorithms + Data Structures = Programs. Prentice-Hall, 1976.

[9] International Mathematical and Statistics Library, Edition 8, June 1980.

[10] Devroye, L. and T. Klincsek. "Average time behavior of distributive sorting algorithms." Computing 26, no. 1, 1981, pp. 1-7.

[11] Baase, S. Computer Algorithms: Introduction to Design and Analysis. Addison-Wesley, 1978.

[12] Huits, M. and V. Kumar. "The practical significance of distributive partitioning sort." Info. Proc. Lett. 8, no. 4, 1979, pp. 168-169.