Behavior Research Methods, Instruments, & Computers
1985, 17 (3), 419-420

Distribution counting as a method for sorting test scores

JOHN K. ELLIS
Carleton University, Ottawa, Ontario, Canada

Distribution counting is a special-purpose algorithm for sorting integers that range from zero to a known maximum (Sedgewick, 1983). In psychology, the algorithm can be used in programs that produce summary results of objective tests, in which scores can range from zero to the number of items, and in similar item-analysis programs. For these types of applications, distribution counting has at least three advantages relative to general-purpose sorting algorithms such as shellsort and quicksort.

The first advantage is simplicity. A BASIC translation of a modified version of Sedgewick's (1983) Pascal code, implemented in Optimized Systems Software, Inc., BASIC XL, on a 6502-based Atari 800 microcomputer, consists of straightforward code (see Listing 1). The algorithm actually sorts data by using cumulative frequencies as array indexes. For example, if the cumulative frequency of the ith score is k, then its sorted position is the kth element of the array Score. Once the ith score has been assigned to its sorted position, its associated cumulative frequency is decremented by 1. That is, there are now (k − 1) unsorted scores with a value less than or equal to the ith score. Thus, the array index for the next score equal to the ith will be (k − 1), the index of the next will be (k − 2), and so on. Repeating this process for each score will sort an array into ascending order.

The version of the algorithm presented here differs from Sedgewick's (1983) in two ways. First, Sedgewick's version requires two passes through the data to read the data and tally their frequencies. The version presented here does both on the same pass. Second, in Sedgewick's version, the data are read into the array Score, sorted into the array Temp, and then read back into the array Score. To eliminate another pass through the data, the present version reads the data into the array Temp and sorts them directly into the array Score. However, if the data are resident in the array Score prior to being sorted (e.g., in an interactive program), they will have to be read into the array Temp before being sorted; and the benefit of this modification will be lost.

The second advantage of the algorithm is that it produces as intermediate results both frequency and cumulative frequency distributions, often required in testing applications. Although the basic version of the algorithm maintains only the cumulative distribution at the termination of a sort, it can be easily modified to maintain both.

The author's mailing address is: Department of Psychology, Ottawa, Ontario, Canada K1S 5J7.

```
100 Rem ,                    LISTING 1
110 Rem
120 Rem
130 Rem ,    DISTRIBUTION COUNTING DEMONSTRATION
140 Rem
150 Rem
160 Read Nitems
170 Read Nscores
180 Data 10,10
190 Rem
200 Rem
210 Dim Freq(Nitems + 1)
220 Dim Score(Nscores)
230 Dim Temp(Nscores)
240 Rem
250 Rem ,    ZERO THE ARRAY FREQ
260 Rem
270   For I = 1 to Nitems + 1
290   Freq(I) = 0
300   Next I
310 Rem
320 Rem ,    READ DATA & TALLY FREQUENCIES
330 Rem
340   For I = 1 To Nscores
350   Read Temp(I)
360   Freq(Temp(I) + 1) = Freq(Temp(I) + 1) + 1
370   Next I
380 Data 8,5,4,7,0,5,6,7,6,3
410 Rem
460 Rem ,    TALLY CUMULATIVE FREQUENCIES
470 Rem
480   For I = 1 To Nitems + 1
490   Freq(I) = Freq(I − 1) + Freq(I)
500   Next I
510 Rem
520 Rem ,    DO ACTUAL SORTING
530 Rem
540   For I = Nscores To 1 Step −1
550   Score(Freq(Temp(I) + 1)) = Temp(I)
560   Freq(Temp(I) + 1) = Freq(Temp(I) + 1) − 1
570   Next I
580 Rem
590 Rem ,    PRINT SORTED SCORES
600 Rem
610   For I = 1 To Nscores
620   Print Score(I)
630   Next I
640 End
```

Listing 1. Distribution counting demonstration.

The third advantage of the algorithm is speed. Distribution counting has an average running time proportional to N; in comparison, quicksort has an average running time proportional to (N log N). Additionally, because initial data order does not affect the number of instructions executed by the distribution counting algorithm, its sorting time will be constant for a given N and maximum value.

As a demonstration, the speed of distribution counting was compared with those of the well-known sorting methods of shellsort and quicksort (the latter being nonrecursive and utilizing insertion sort). Five samples each,

of arrays of 25, 50, 100, and 200 pseudorandom integers (0-100), were sorted by each method. Before being sorted, each sample was read into the array Score, with the initial data order kept the same for each sorting method.

The demonstration results (see Table 1) show that distribution counting is the fastest method, except for the smallest array size. In general, distribution counting gains advantage as the ratio of array size to the range of the integers being sorted increases. As expected, quicksort was faster than shellsort for all array sizes.

Table 1
Mean Sorting Time in Seconds

	Array Size			
Method	25	50	100	200
Shellsort	1.3	3.2	7.8	19.5
Quicksort	1.1	2.6	5.7	13.2
Distribution Counting	1.9	2.5	4.0	6.9

The particulars of an application determine the practical value of these results. However, it is clear that when the extra memory required for the array Temp is not prohibitive, distribution counting is worth considering as a sorting method for test-scoring and item-analysis programs.

REFERENCE

SEDGEWICK, R. S. (1983). *Algorithms*. Reading, MA: Addison-Wesley.

(Revision accepted for publication March 1, 1985.)