Soughts of sorts

I read with great interest your recent Mike Liardet series review of Donald Knuth's "The Art of Computer Programming". However, in view of its rather spectacular performance against other sorts and because I had never even heard of the distribution sort before, I was more than a little disappointed that Mr Liardet dismissed it with the comment that it was "not universally applicable".

One of the monotonously repetitive tasks of my computer (and, I'm sure, of many of your readers') is the sorting of string arrays in which the keys are the string elements themselves. I'm therefore constantly on the lookout for a faster sort routine and immediately determined to try to harness the distribution sort.

After a full day of hacking (and tearing my hair out!) I finally settled on the following algorithm as being probably optimal:

1 the use of the distribution sort as a presort of a randomly-ordered string array to arrange the array so that all string elements would be clustered with others sharing the same initial (naturally in ascending order); and then

2 the use of an "intelligent" bubble sort to arrange each cluster of elements in lexicographic order.

The following program, written on a Commodore 64, generates a randomly-ordered array of strings, each of 30 characters' length, and applies this distribution/bubble sort technique to it. The TI$ reference is, of course, the onboard C64 timer; interested readers using other computers will have to work out their own timing devices.

I have chosen 300 as the size of the array because, as the following Benchmark chart shows, that size appears to be the break-even point between my sort and the old faithful Quicksort. However, my sort has the advantage that, if the randomly-ordered array just happens to be more or less truly ordered to begin with, it will perform considerably faster, whereas — as everyone knows — the Quicksort will be disastrously slower in such a circumstance.

For instance, one Quicksort run I performed on a 500-element array took 9 minutes 45 seconds as opposed to the mean 4 minutes 50 seconds shown in the chart shown, leading me to suspect that that particular randomly-ordered array was not as random as it might have been!

I do hope these observations will be of interest to some of your readers.

K Riordan

```
 1 N=300 : DIM A$(N),C(26) : FOR X=1 TO N : FOR
   Y=1 TO 30
 2 Z=INT(RND(0)*26)+65 : A$(X)=A$(X)+CHR$(Z) :
   NEXT : NEXT
 3 TI$="000000" : PRINT TI$" SORTING . . ."
 4 FOR J=1 TO N : A=ASC(A$(J))-64 : C(A)=C(A)+1
   : NEXT
 5 FOR J=2 TO 26 : C(J)=C(J)+C(J-1) : NEXT :
   R=N+1
 6 R=R-1 : IF R=0 THEN 13
 7 A=ASC(A$(R))-64 : IF C(A)<R THEN 6
 8 IF C(A)=R THEN C(A)=C(A)-1 : GOTO 6
 9 A$=A$(R) : J=C(A) : C(A)=C(A)-1
10 B$=A$(J) : A=ASC(B$)-64 : K=C(A) : C(A)=C(A)-1
   : A$(J)=A$ A$=B$ : J=K : IF J<>R THEN 10
11 A$(J)=A$ : GOTO 6
12 REM *** NOW FOR THE "INTELLIGENT" BUBBLE
   SORT! ***
13 FOR J=1 TO N-1 : FOR K=J+1 TO N
14 IF ASC (A$(J))<ASC(A$(K)) THEN K=N : GOTO 16
15 IF A$(J)>A$(K) THEN A$=A$(J) : A$(J)=
   A$(K)=A$(K)=A$
16 NEXT : NEXT : PRINT TI$" SORTED!" : END
```

The Benchmarks:

# OF ELEMENTS	MY SORT	QUICKSORT
50	00' 05"	00' 08"
100	00' 14"	00' 20"
150	00' 24"	00' 41"
200	00' 45"	00' 55"
250	01' 09"	01' 17"
300	01' 45"	01' 45"
500	07' 05"	04' 50"

APC Jun 85 6(6) p 171 and 173.