```
      SUBROUTINE SORT(X, N)
C     SORT INCREASING, BY INTERCHANGE
      REAL X(N)
      IF (N .LT. 2) RETURN
      DO 20 I = 2, N
         DO 10 J = 1, I
            IF (X(I) .GE. X(J)) GOTO 10
               SAVE = X(I)
               X(I) = X(J)
               X(J) = SAVE
   10    CONTINUE
   20 CONTINUE
      RETURN
      END
```

This has about half as many lines of code as the "efficient" sort, and is simple enough that comments seem superfluous. We have coded it as a subroutine, a more likely usage. (Notice the immediate return if N is less than 2.)

How much faster is the "efficient" program than the simple-minded one? We eliminated the I/O statements from the former and made it into a subroutine, so we could directly compare sort times without I/O overhead. Then we sorted arrays of uniformly distributed random numbers (several arrays of each size). Here are some run times, in milliseconds:

size	"efficient"	simple	ratio
10	1	1	1.0
50	22	19	1.15
300	850	670	1.25
2000	38500	29200	1.3

As we might have anticipated, complexity again loses out to simplicity: not only has carefully-tailored code produced a 15 to 30 percent *increase* in run time, but the ratio appears to be getting worse as the size goes up.

Keep it simple to make it faster.

Although the simpler code is faster, it is still time-consuming for larger arrays — 30 seconds to sort 2000 numbers is extravagant *if it is done often.* (Done infrequently, it is probably irrelevant; the programmer time needed to make a noticeable improvement in speed is certainly more valuable than a few minutes of machine time.)

How can we really speed it up? Fundamental improvements in performance are most often made by algorithm changes, not by tuning. Let us demonstrate.

It is well known in the sorting business that the interchange sort is suitable only for sorting a handful of items. Here is a simple version of a better procedure, known as the Shell sort (after D. L. Shell). Conceptually it is similar to the "efficient" sort we began with, and certainly no more complicated.