```
      DIMENSION X(300)
C     READ NUMBERS TO BE SORTED.
      READ 1,N,(X(I),I=1,N)
    1 FORMAT(I3/(F5.1))
C     INITIALIZE TO MAKE N-1 COMPARISONS ON FIRST PASS.
      K=N-1
C     INITIALIZE TO BEGIN COMPARISONS WITH THE FIRST 2 NUMBERS.
    6 J=1
C     L IS USED TO RECORD THE FACT THAT AN INTERCHANGE OCCURS.
   19 L=0
C     MAKE COMPARISONS.
      DO 2 I=J,K
      IF(X(I)-X(I+1)) 2,2,3
C     AN INTERCHANGE IS TO TAKE PLACE.
C     IS THIS THE FIRST INTERCHANGE.
    3 IF(L) 20,21,20
C     RECORD POINT OF FIRST INTERCHANGE LESS ONE POSITION.
   21 J1=I-1
C     MAKE INTERCHANGE.
   20 SAVE=X(I)
      X(I)=X(I+1)
      X(I+1)=SAVE
C     RECORD POINT OF LAST INTERCHANGE (ACTUALLY ALL INTERCHANGES).
      L=I
    2 CONTINUE
C     DETERMINE IF NUMBERS ARE IN SEQUENCE.
      IF(L) 8,9,8
C     NUMBERS ARE NOT YET IN SEQUENCE.  SET DO PARAMETERS.
    8 K=L
C     DO NOT WANT TO START AT ZERO SO TEST J1 FOR VALUE OF 0.
      IF(J1) 6,6,7
    7 J=J1
      GO TO 19
    9 PRINT 16,N
      ... print numbers, etc.
```

In Chapter 5 we mentioned the perils of making the user specify the number of data points to be input rather than letting the machine do the counting. We have also discussed how this type of code fails when N is less than two. Similarly, we have often pointed out that arithmetic IF's are inadvisable, for they are less clear to the reader (Quickly! Does it sort up or down?), and always add the possibility of arithmetic overflow or underflow. And the hodge-podge of statement numbers makes it unnecessarily difficult to find one's way around the code.

But for now our primary subject is "efficiency." Inspection reveals that this sort program is carefully coded to squeeze most of the possible speed out of the basic algorithm. A switch L determines whether the table has been sorted in less than the maximum N–1 passes, so an early exit can be taken. The index J increases to skip over elements known to be already in order at the beginning of the array. The upper index K decreases over those in order at the end. The programmer has carefully avoided the trap of letting J become zero. And there are plenty of comments to explain what is going on. All in all, this should be a marked improvement over a basic no-bells-and-whistles version.

Let us put that hypothesis to the test, by constructing another sort program and comparing run times on identical data. Here is our interchange sort, absolutely devoid of frills. We do not even bother to eliminate comparisons between an element and itself.