```
    5 IF (KA .EQ. 0) GOTO 7
        KR = MOD(KB,KA)
        KB = KA
        KA = KR
        GOTO 5
    7 PRINT 102, KB
```

This is of course a DO-WHILE — the division is repeated while KA is not zero. And notice that it can be done zero times.

As another small instance of the same thing, consider

```
      DO 10 I=1,M
      IF(BP(I)+1.0)19,11,10
   11 IBN1(I) = BLNK
      IBN2(I) = BLNK
      GO TO 10
   19 BP(I) = -1.0
      IBN1(I) = BLNK
      IBN2(I) = BLNK
   10 CONTINUE
```

If BP(I) is less than or equal to −1, this excerpt will set BP(I) to −1 and put blanks in IBN1(I) and IBN2(I). The code uses a hard-to-read Fortran arithmetic IF that branches three ways, two almost-duplicated pieces of code, two extra labels and a GOTO, all to avoid setting BP(I) to −1 if it is already.

There is no need to make a special case. Write the code so it can be read:

```
      DO 10 I = 1, M
         IF (BP(I) .GT. -1.0) GOTO 10
            BP(I) = -1.0
            IBN1(I) = BLNK
            IBN2(I) = BLNK
   10 CONTINUE
```

Interestingly enough, our version will be more "efficient" on most machines, both in space and in time: although we may occasionally reset BP(I) unnecessarily, we do less bookkeeping. What did concern with "efficiency" in the original version produce, besides a bigger, slower, and more obscure program?

Make sure special cases are truly special.

Let us turn to sorting, an area where efficiency *is* important in practice. Here is an interchange sort: