efficient sorting routine, but more sophisticated sorts are much harder to understand than the one we have used. Whether or not you should consider a better sort routine depends on the number of items to be sorted. The 'time complexity' of a bubble sort such as ours is n^2. In other words, the time taken for the data to be sorted increases as the square of the number of items being sorted. If two items took four milliseconds to sort, four items would take 16 milliseconds, 50 items would take two and a half seconds and 1,000 items would take more than 16 minutes. A wait of two or three seconds might be perfectly acceptable during the use of a program like ours, but a wait of a quarter of an hour certainly wouldn't be.

The way this program has been written allows a maximum of only 50 records, so unacceptable delays during sorts should not be a problem. Later in the course, however, we shall outline some of the techniques that can be used to create dynamic files that can grow to almost any size. If you do attempt such a modification to the program, a more advanced sort routine would be one of the first problems to be tackled.

The data items being sorted are the character strings in MODFLD$(L) and MODFLD$(L+1). Records are swapped only if MODFLD$(L) is greater than MODFLD$(L+1), and the index field (which is not being used at present) is updated in lines 11490 and 11570. Every time two records have been swapped, the variable S (to indicate that a swap has taken place) is set to 1. When the sorting routine reaches line 11290 it checks the value of S and branches back to compare all the records again. When all the records are in order, the value of S will be left at 0 and the routine will be terminated after the value of RMOD has been reset to 0.

The EXPROG routine (referred to as *EXPROG* in the program listing) begins at line 11000. It starts by checking to see if any record has been modified during the current execution of the program (line 11050: IF RMOD=0 THEN RETURN). If there has been no modification of the file, there will be no need to save again, so the routine RETURNs to the main program. This will take us back to line 100, which checks the value of CHOI. If CHOI has a value of 9 (as it would if *EXPROG* is being executed) the main program simply goes on to the END statement in line 110.

If the program finds that RMOD is 1 in line 11050 it means that one or more records have been modified in some way and that there is a chance that they are no longer in order. This being so, the *EXPROG* routine calls the sort routine (line 11070) and then, after all the records have been sorted, saves them onto tape or disk.

The save routine (*SAVREC*) is called in line 11090 and the routine starts at line 12000. *SAVREC*, in the main listing, is written in Microsoft BASIC, so it is important to bear in mind that the details of file-handling vary from one version of BASIC to another (see 'Basic Flavours'). Line 12030 opens the ADBK.DAT data file and assigns the channel number #1 for the operation. Line 12050 sets the limits for the loop that counts through all the records in the file. The upper limit is SIZE-1, not SIZE, because the SIZE variable always has a value one greater than the number of valid records in the file (so that if a new record is added, it will not be written over an existing record).

The format of lines 12060 and 12070 is particularly noteworthy. Each field is separated by a ",", which is also sent to the file. This comma is required by most versions of BASIC because INPUT# and PRINT# work in the same way as the ordinary INPUT and PRINT statements. Consider the statement INPUT X,Y,Z. This would expect an input from the keyboard such as 10,12,15<CR>, which would assign 10, 12 and 15 to X, Y and Z respectively. Without the commas, the INPUT statement would not be able to tell where each data item ended and would assign all the data to the first variable. Similarly, the INPUT# statement (in most BASICS) would not be able to tell where each data file record ended and would try to fill each string variable with as much data as could be fitted in. Since in most BASICS string variables can hold up to 255 characters, the data in the data file would soon all be assigned long before the FOR L = 1 TO SIZE-1 loop had terminated. This would result in an INPUT PAST END error message (which indicates that an INPUT statement was issued after all the data has been exhausted) and the string variables (such as NAMFLD$(x)) containing far more data than they should.

Once all the records have been stored in the data file, from L=1 TO SIZE - 1, *SAVREC* RETURNs to line 90 in the main program. Line 100 checks the value of CHOI to see if the last operation was *EXPROG* or not. If it was 9 (save and exit), the program goes on to the END statement in line 110. If CHOI has any other value, the program jumps back to *CHOOSE* and allows the user to select another option again.

As a final footnote, we should mention the *FLSIZE* routine that starts at line 12500. This is offered as a possible alternative to the statement in line 1510. As presented, the program depends on the presence of an end-of-file function: IF EOF(1) = -1 THEN LET L = 50. All BASICS have some way of indicating that the end of a file has been reached, either with a special function such as EOF(x) or a PEEK to a special memory location. The *FLSIZE* routine at line 12500 is offered as a suggestion if an EOF function is not available, in which case line 1510 would need to be replaced by GOSUB 12500.

Basic Flavours

SPECTRUM

Before running the address book program you must create on tape the name-field file. The following program will achieve this.

```
10 REM PROGRAM TO CREATE NFLD FILE ON
   TAPE
20 DIM Z$(1,30)
30 LET Z$(1)="@FIRST"
40 SAVE "NFLD" DATA Z$( )
50 STOP
```