1 SUMMARY

To **sort a table of fixed length entries** into ascending or descending order, **sorting on a key field** within each entry. The key field is assumed to contain non-numeric information and the length of the key and its position in the record must be the same for each sort of key.

The entries in the table are not moved but rather an index array is returned specifying the required ascending or descending order. The initialization of the index array must be done by the user and this gives the user control over subsorting on

ATTRIBUTES — **Version:** 1.0.0. **Remark:** Excellent for sorting text type information. to locate specific entries after the table is sorted. **Types:** KB10A. **Calls:** IC02A. **Original date:** May 1969. **Origin:** K.Moody, IBM.

2 HOW TO USE THE PACKAGE

2.1 The argument list and calling sequence

```
CALL KB10A(TAB,INDX,N,KPOS,KLEN,LREC,IWORK)
```

TAB is a CHARACTER array containing the table to be sorted. It is regarded by the subroutine as a continuous string of one-byte fields, subdivided according to the entry length LREC. This argument is not altered by the subroutine. **N.B.** the entries in TAB are not physically re-ordered by the subroutine.

INDX is an INTEGER array of length at least N. For a simple sort into ascending or descending order the user must initialize the array so that INDX(I)=I for I=1,N. On return from the subroutine the array will give the required ordering, i.e., the I th record in the ordered table will be located as the INDX(I) th record in TAB. If subsorting on several fields is being done the contents of the array should not be altered by the user between calls of KB10A and the junior fields should be sorted first.

N is an INTEGER variable. The absolute value of N gives the number of records in the table in TAB. If N>0 the table is sorted into ascending order, if N<0 into descending order. This argument is not altered by the subroutine.

KPOS is an INTEGER variable set by the user to the byte position of the first byte of the key in each entry. The byte positions in an entry are numbered 1,2,... This argument is not altered by the subroutine.

KLEN is an INTEGER variable set by the user to the length This argument is not altered by the subroutine. of the key in bytes, maximum 256.

LREC is an INTEGER variable set by the user to the length of a table entry in bytes. **N.B.** if TAB is a two-dimensional array LREC is set to the first dimension times the word length of TAB. This argument is not altered by the subroutine.

IWORK is an INTEGER workspace array of length at least N.

2.2 Common

The subroutine contains a common area which the user may wish to reference.

```
COMMON /KB10B/ LP, MERR
```

LP is an INTEGER variable which specifies the Fortran stream number to be used for error messages. The default value is 6. To suppress the printing of error messages set LP to zero.

MERR is an INTEGER variable which is always set by the subroutine to indicate success or failure. On exit from the

subroutine MERR will take one of the following values.

0 successful entry.
1 KPOS ≤ 0, KLEN ≤ 0, or LREC < 0.
2 KLEN > 256.
4 N < 2.

3 GENERAL INFORMATION

Use of Common: uses common area KB10B.

Workspace: see argument IWORK.

Other subroutines: calls IC02A.

Input/Output: error messages, see § 2.2.

4 METHOD

A merge sort is performed. After k iterations of the main loop, the subsets $[1,\ 2^k]$, $[1+2^k,\ 2\times2^k]$, $[1+2\times2^k,\ 3\times2^k]$, ... are each in order. In the next iteration, merges are performed between the first two subsets, the next two subsets, ... Thus the total number of comparisons is approximately $\frac{1}{2}n\log_2 n$.

5 EXAMPLE OF USE

Suppose we have stored in a CHARACTER*8 array TABLE, (dimensioned TABLE(10,100)), 74 records of the form:

bytes 1 to 8: telephone no.

bytes 9 to 24: name

bytes 25 to 40: town

bytes 41 to 80: local address

and it is required to sort these so that the names are in alphabetical order and to print out the ordered table. The code for this might be as shown in figure 1.

If it was required to group the names according to towns, with both towns and names within town groups to be in alphabetical order, we would need to do a further sort but leaving the contents of INDX unchanged, then the code in figure 1 could be followed by the code in figure 2. At the end of this INDX would define the required grouping.

The code to sort the table by names.

```
C      THE TABLE TO BE SORTED
       CHARACTER*8 TABLE(10,100)
C      INDX WILL CONTAIN THE ORDERING
       INTEGER INDX(100)
         -
C      SET ARGUMENT VALUES
C      NUMBER OF EMTRIES POSITIVE FOR ASCENDING ORDER.
       N=74
C      SPECIFY NAME FIELD TO BE THE SORT KEY.
       KPOS=9
C      LENGTH OF NAME FIELD.
       KLEN=16
C      FIRST DIMENSION OF TABLE TIMES ITS WORD LENGTH.
       LREC=80
C      INITIALISE INDEX ARRAY.
       DO 1 I=1,N
     1 INDX(I)=I
C      SORT THE TABLE
       CALL KB10A(TABLE,INDX,N,KPOS,KLEN,LREC)
C      PRINT ORDERED TABLE - ORDERING GIVEN IN INDX
       DO 2 I=1,N
     2 WRITE(6,3) (TABLE(J,INDX(I)),J=1,10)
     3 FORMAT(5X,10A8)
```

The extra code to do a further sort on the towns.

```
C      SWITCH TO TOWN FIELD AS KEY
C      SPECIFY TOWN FIELD AS SORT KEY.
       KPOS=25
C      LENGTH OF TOWN FIELD.
       KLEN=16
C      SORT ON TOWN NAMES LEAVING INDX ARRAY UNCHANGED
       CALL KB10A(TABLE,INDX,N,KPOS,KLEN,LREC)
         -
```