1 SUMMARY

To **sort** n **numbers from an array of** m **numbers,** $n < m$. Options are provided for sorting either the first n smallest or the first n largest numbers either in terms of their algebraic values or their absolute values. An index giving the ordering of the first n numbers and the n ordered numbers themselves are returned but the physical order of the original array of numbers is not altered.

Repeated sequential searches or a total sort are alternative approaches to this problem. The break even points for these methods and that of `KB21` are discussed in §4.

ATTRIBUTES — **Version:** 1.0.0. **Types:** `KB21A; KB21AD; KB21AI.` **Calls:** `FD05`. **Original date:** January 1993. **Origin:** M.J.Hopper, Harwell. **Remark:** Originally `KB11` with integer workspace added.

2 HOW TO USE THE PACKAGE

2.1 The argument list and calling sequence

(a) To sort n numbers from m in terms of their **algebraic values.**

The single precision version:

```
CALL KB21A(M,N,A,IND,W,IW)
```

The double precision version:

```
CALL KB21AD(M,N,A,IND,W,IW)
```

The integer version:

```
CALL KB21AI(M,N,A,IND,W,IW)
```

(b) To sort n numbers from m in terms of their **absolute values.**

The single precision version:

```
CALL KB21B(M,N,A,IND,W,IW)
```

The double precision version:

```
CALL KB21BD(M,N,A,IND,W,IW)
```

The integer version:

```
CALL KB21BI(M,N,A,IND,W,IW)
```

`M` is an `INTEGER` variable set by the user to m the number of numbers in the array. This argument is not altered by the subroutine.

`N` is an `INTEGER` variable set by the user so that the absolute value of `N` equals n the number of numbers to be sorted from the total array of m numbers. If `N≥0` the first n smallest numbers (in ascending order) are found; if `N<0` the n largest numbers (in descending order) are found.

`A` is an **array** set by the user to the array of numbers to be sorted. It can be `REAL, DOUBLE PRECISION` or `INTEGER` provided the version of the subroutine is chosen to correspond with the type. This argument is not altered by the subroutine.

`IND` is an `INTEGER` array of length n in which the subroutine will return the positions in the array `A` of the first n ordered elements, i.e. the required numbers will be in `A(IND(J)), J=1,N.`

`W` is an **array** which must be declared the same Fortran type as the array `A`. It is used as workspace and must be of length at least n+2k words long, where

$$k = \sqrt{\frac{(n-1)m}{n}} + 1.$$

On return the first n elements of `W` will contain the n ordered numbers sorted from `A`.

`IW` is an `INTEGER` array of length at least k (see `W`) to be used by the routine as workspace.

3 GENERAL INFORMATION

Use of Common: uses a private common block `KB21D/DD/DI`.

Workspace: provided by the user in `W` and `IW`.

Other subroutines: calls private routines `KB21C/E/F/G/H`, `KB21CD/ED/FD/GD/HD` and `KB21CI/EI/FI/GI/HI`.

Input/Output: none.

Restrictions: $m \geq n$, no diagnostic is printed if this restriction is violated and $m = max(m,n)$ is used.

4 METHOD

The array of m numbers is divided into k sections and the optimal element in each section is recorded. These k elements are then scanned to locate the overall optimal element; it is recorded and blanked out by setting it to the largest possible number (or smallest depending on the type of sort). The next optimum element in that section is found and included in the set of k elements which are then scanned for the next overall optimum element. The process is repeated until the n optimum elements have been found. The total number of comparisons required to find the n elements is $m + (n-1)\frac{m}{k} + nk$ and the optimal value for k is

$$k = \sqrt{\left\{\frac{(n-1)m}{n}\right\}}.$$

There are two other methods which could be used to sort n numbers from m when n is either very small compared with m or of the same order as m.

(i) if n is very small, 1 or 2 say; repeated sequential searches could be used and calls to `KB21`, and the overhead this carries with it, avoided. This method will get expensive as n increases. The break even point with `KB21` is at about $n = 3$ for $m < 100$ and going down to $n = 1$ as m increases.

(ii) if n is of the same order of m a total sort will be worthwhile. The number of comparisons required by `KB21` as n gets large is very approximately $\beta(m + 2n\sqrt{m})$ and the number for its competitor `KB05` will be of the order of $\alpha m \log_2(m)$ for a total sort. Assuming rough estimates for the factors α and β the break even point will behave like

$$\left(\frac{\log_2(m)}{\sqrt{m}}\right) \times m.$$

For m around 100 it comes out in practice to be about $n = \frac{1}{2}m$ and for m about 1000 goes down to about $\frac{1}{3}m$.