Chapter m01 – Sorting

1. Scope of the Chapter

This chapter provides functions for sorting data. Most of the functions can handle data of arbitrary type which is sorted according to a user-supplied comparison function.

2. Background

The general problem may be defined as follows. We are given n items of data

$$R_1, R_2, \ldots, R_n.$$

Each item R_i contains a key K_i which can be ordered relative to any other key according to some specified criterion (for example, ascending numeric value). The problem is to determine a permutation

$$p(1), p(2), \ldots, p(n)$$

which puts the keys in order:

$$K_{p(1)} \leq K_{p(2)} \leq \ldots \leq K_{p(n)}$$

Sometimes we may wish actually to *rearrange* the items so that their keys are in order; for other purposes we may simply require a table of *indices* so that the items can be referred to in sorted order; or yet again we may require a table of *ranks*, that is, the positions of each item in the sorted order.

For example, given the single-character items, to be sorted into alphabetic order:

E B A D C

the indices of the items in sorted order are

3 2 5 4 1

and the ranks of the items are

5 2 1 4 3

Indices may be converted to ranks, and vice versa, by simply computing the inverse permutation.

The items may consist solely of the key. On the other hand, the items may contain additional information, and then there may be many distinct items with equal keys, and it may be important to preserve the original order among them: if this is achieved, the sorting is called *stable*.

The *Quicksort* algorithm, used by nag_quicksort (m01csc), is not stable in this sense; hence an alternative function, nag_stable_sort (m01ctc), is provided which does perform a stable sort, but requires more internal workspace, and may be slower.

3. Available Functions

3.1. Functions which rearrange the data into sorted order

Chain sort of linked list of items of arbitrary data type	`m01cuc`
Quicksort of vector of type double	`m01cac`
Quicksort of vector of arbitrary data type	`m01csc`
Stable sort of vector of arbitrary data type	`m01ctc`

3.2. Function which determines the ranks of the data, leaving it unchanged

Rank a vector of arbitrary data type	`m01dsc`

3.3. Function which rearranges the data according to pre-determined ranks

Rearrange a vector of arbitrary data type	`m01esc`

3.4. Utility functions

Invert a permutation, converting a rank vector to an index vector or vice versa	`m01zac`
Search a vector of arbitrary data type for a match to a given key	`m01fsc`