NAG Library Chapter Introduction

m01 – Sorting and Searching

Contents

1 Scope of the Chapter

This chapter is concerned with sorting numeric or character data. It handles only the simplest types of data structure and it is concerned only with **internal** sorting – that is, sorting a set of data which can all be stored within the program.

If you have large files of data or complicated data structures to be sorted you should use a comprehensive sorting program or package.

2 Background to the Problems

2.1 Sorting

The usefulness of sorting is obvious (perhaps a little too obvious, since sorting can be expensive and is sometimes done when not strictly necessary). Sorting may traditionally be associated with data processing and non-numerical programming, but it has many uses within the realm of numerical analysis. For example, within the NAG C Library, sorting is used to arrange eigenvalues in ascending order of absolute value, in the manipulation of sparse matrices, and in the ranking of observations for nonparametric statistics.

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 (each item may simply be a number). On the other hand, the items may contain additional information (for example, each item may be an eigenvalue of a matrix and its associated eigenvector, the eigenvalue being the key). In the latter case 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**').

There are a number of ingenious algorithms for sorting. For a fascinating discussion of them, and of the whole subject, see Knuth (1973).

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.

2.2 Searching

Searching is a process of retrieving data stored in a computer's memory.

The general problem may be defined as follows:

> We are given n items of data that have been sorted and a sought-after item x. Each item contains a key. The problem is to find which item has x as its key.
>
> We may be interested in different information gained from the search. We may wish to know if item x was or was not found or the position of the item that was found.

There are a number of different search algorithms. For more on the subject, see Knuth (1973) and Wirth (2004).

3 Functionality Index

Determined ranks of the data,
- unchanged,
  - vector,
    - arbitrary types .... nag_rank_sort (m01dsc)

Pre-determined ranks of the data,
- vector,
  - arbitrary types .... nag_reorder_vector (m01esc)

Searching (i.e., exact match or the nearest lower value):
- binary search,
  - vector,
    - integer numbers .... nag_search_int (m01nbc)
    - null terminated strings .... nag_search_char (m01ncc)
    - real numbers .... nag_search_double (m01nac)
- search for a match to a given key .... nag_search_vector (m01fsc)

Service functions,
- invert a permutation (ranks to indices or vice versa) .... nag_make_indices (m01zac)

Sorting (i.e., rearranging into sorted order):
- chain sort,
  - linked list of items,
    - arbitrary types .... nag_chain_sort (m01cuc)
- quick sort,
  - vector,
    - arbitrary types .... nag_quicksort (m01csc)
    - real numbers .... nag_double_sort (m01cac)
- stable sort,
  - vector,
    - arbitrary types .... nag_stable_sort (m01ctc)

4 Auxiliary Functions Associated with Library Function Arguments

None.

5 Functions Withdrawn or Scheduled for Withdrawal

None.

6 References

Knuth D E (1973) *The Art of Computer Programming (Volume 3)* (2nd Edition) Addison–Wesley

Wirth N (2004) *Algorithms and Data Structures* 35–36 Prentice Hall