nag_quicksort (m01csc)

1. Purpose

nag_quicksort (m01csc) rearranges a vector of arbitrary type objects into ascending or descending order.

2. Specification

```
#include <nag.h>
#include <nag_stddef.h>
#include <nagm01.h>

void nag_quicksort(Pointer vec[], size_t n, size_t size, ptrdiff_t stride,
     Integer (*compare)(const Pointer, const Pointer), Nag_SortOrder order,
     NagError *fail)
```

3. Description

nag_quicksort sorts a set of n data objects of arbitrary type, which are stored in the elements of an array at intervals of length **stride**. The function may be used to sort a column of a two-dimensional array. Either ascending or descending sort order may be specified.

nag_quicksort is based on Singleton's implementation of the 'median-of-three' Quicksort algorithm, Singleton (1969), but with two additional modifications. First, small subfiles are sorted by an insertion sort on a separate final pass, Sedgewick (1978). Second, if a subfile is partitioned into two very unbalanced subfiles, the larger of them is flagged for special treatment: before it is partitioned, its end-points are swapped with two random points within it; this makes the worst case behaviour extremely unlikely.

4. Parameters

vec[ ]
: Input: the array of objects to be sorted.
Output: the objects rearranged into sorted order.

n
: Input: the number, n, of objects to be sorted.
Constraint: $\mathbf{n} \geq 0$.

size
: Input: the size of each object to be sorted.
Constraint: $\mathbf{size} \geq 1$.

stride
: Input: the increment between data items in **vec** to be sorted.
Note: if **stride** is positive, **vec** should point at the first data object; otherwise **vec** should point at the last data object.
Constraint: $|\mathbf{stride}| \geq \mathbf{size}$.

compare
: User-supplied function: this function compares two data objects. If its arguments are pointers to a structure, this function must allow for the offset of the data field in the structure (if it is not the first).
The function must return:
-1 if the first data field is less than the second,
0 if the first data field is equal to the second,
1 if the first data field is greater than the second.

order
: Input: Specifies whether the array is to be sorted into ascending or descending order.
Constraint: **order** = **Nag_Ascending** or **Nag_Descending**.

fail
The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

NE_INT_ARG_LT
On entry, **n** must not be less than 0: **n** = ⟨*value*⟩.
On entry, **size** must not be less than 1: **size** = ⟨*value*⟩.
The absolute value of **stride** must not be less than **size**.

NE_INT_ARG_GT
On entry, **n** must not be greater than ⟨*value*⟩: **n** = ⟨*value*⟩.
On entry, **size** must not be greater than ⟨*value*⟩: **size** = ⟨*value*⟩.
These parameters are limited to an implementation-dependent size which is printed in the error message.

NE_2_INT_ARG_LT
On entry, |**stride**| = ⟨*value*⟩ while **size** = ⟨*value*⟩. These parameters must satisfy |**stride**— ≥ **size**.

NE_BAD_PARAM
On entry, parameter **order** had an illegal value.

NE_ALLOC_FAIL
Memory allocation failed.

6. Further Comments

The average time taken by the function is approximately proportional to $n \log n$. The worst case time is proportional to n^2 but this is extremely unlikely to occur.

6.1. References

Maclaren N M (1985) *Comput. J.* **28** 446.
Sedgewick R (1978) Implementing Quicksort programs *Commun. ACM* **21** 847–857.
Singleton R C (1969) An efficient algorithm for sorting with minimal storage: Algorithm 347 *Commun. ACM* **12** 185–187.

7. See Also

nag_stable_sort (m01ctc)

8. Example

The example program reads a two-dimensional array of numbers and sorts the second column into ascending order.

8.1. Program Text

```
/* nag_quick_sort(m01csc) Example Program
 *
 * Copyright 1990 Numerical Algorithms Group.
 *
 * Mark 2 revised, 1992.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nag_stddef.h>
#include <nagm01.h>

#ifdef NAG_PROTO
static Integer compare(const Pointer a, const Pointer b)
#else
     static Integer compare(a, b)
```

```
    Pointer a, b;
#endif
{
  double x = *((double *)a);
  double y = *((double *)b);
  return (x<y ? -1 : (x==y ? 0 : 1));
}

#define MMAX 20
#define NMAX 20

main()
{
  double vec[MMAX][NMAX];
  Integer i, n, m, j, k;
  static NagError fail;

  fail.print = TRUE;
  /* Skip heading in data file */
  Vscanf("%*[^\n]");
  Vprintf("m01csc Example Program Results\n");
  Vscanf("%ld", &m);
  Vscanf("%ld", &n);
  Vscanf("%ld", &k);
  if (m>0 && m<MMAX && n>0 && n<NMAX && k>0 && k<n)
    {
      for (i=0; i<m; ++i)
        for (j=0; j<n; ++j)
          Vscanf("%lf",&vec[i][j]);
      m01csc((Pointer) &vec[0][k-1], (size_t) m, sizeof(double),
             (ptrdiff_t)(sizeof(double)*NMAX), compare, Nag_Ascending, &fail);
      if (fail.code != NE_NOERROR)
        exit (EXIT_FAILURE);
      Vprintf("\nMatrix with column %ld sorted \n", k);
      for (i=0; i<m; ++i)
        {
          for (j=0; j<n; ++j)
            Vprintf(" %8.1f ", vec[i][j]);
          Vprintf("\n");
        }
      exit(EXIT_SUCCESS);
    }
  else
    {
      Vfprintf(stderr, "Data error: program terminated.\n");
      exit (EXIT_FAILURE);
    }
}
```

8.2. Program Data

```
m01csc Example Program Data
12 3 2
      6.0       5.0       4.0
      5.0       2.0       1.0
      2.0       4.0       9.0
      4.0       9.0       6.0
      4.0       9.0       5.0
      4.0       1.0       2.0
      3.0       4.0       1.0
      2.0       4.0       6.0
      1.0       6.0       4.0
      9.0       3.0       2.0
      6.0       2.0       5.0
      4.0       9.0       6.0
```

8.3. Program Results

```
m01csc Example Program Results

Matrix with column 2 sorted
      6.0       1.0       4.0
      5.0       2.0       1.0
      2.0       2.0       9.0
      4.0       3.0       6.0
      4.0       4.0       5.0
      4.0       4.0       2.0
      3.0       4.0       1.0
      2.0       5.0       6.0
      1.0       6.0       4.0
      9.0       9.0       2.0
      6.0       9.0       5.0
      4.0       9.0       6.0
```