nag_rank_sort (m01dsc)

1. Purpose

nag_rank_sort (m01dsc) ranks a vector of arbitrary data type objects in ascending or descending order.

2. Specification

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

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

3. Description

nag_rank_sort ranks a set of n data objects of arbitrary type, which are stored in the elements of an array at intervals of length **stride**. The ranks are in the range 0 to $n - 1$.

Either ascending or descending ranking order may be specified.

nag_rank_sort uses a variant of list merging as described by Knuth (1973).

4. Parameters

vec[ ]
Input: the array of objects to be ranked.

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

stride
Input: the increment between data items in **vec** to be ranked.
Note: if **stride** is positive, **vec** should point at the first data object; otherwise **vec** should point at the last data object.
It should be noted that |**stride**| must be greater than or equal to size_of (data objects), for correct ranks to be produced. However, the code performs no check for violation of this constraint.
Constraint: $|\mathbf{stride}| > 0$.

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 ranked into ascending or descending order.
Constraint: **order** = **Nag_Ascending** or **Nag_Descending**.

ranks[n]
Output: the ranks of the corresponding data elements in **vec**.

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*⟩.

NE_INT_ARG_GT
On entry, **n** must not be greater than ⟨*value*⟩: **n** = ⟨*value*⟩.
On entry, |**stride**| must not be greater than ⟨*value*⟩: **stride** = ⟨*value*⟩.

These parameters are limited to an implementation-dependent size which is printed in the error message.

NE_INT_ARG_EQ
On entry, **stride** must not be equal to 0: **stride** = ⟨*value*⟩.

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

6. Further Comments

The time taken by the function is approximately proportional to $n \log n$.

6.1. References

Knuth D E (1973) *The Art of Computer Programming (Vol 3, Sorting and Searching)* Addison-Wesley.

7. See Also

None.

8. Example

The example program reads a list of real numbers and ranks them into ascending order.

8.1. Program Text

```
/* nag_rank_sort(m01dsc) Example Program
 *
 * Copyright 1990 Numerical Algorithms Group.
 *
 * Mark 4, 1996.
 */

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

#define N_MAX 50

#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));
}

main()
{
  double vec[N_MAX];
  ptrdiff_t step;
```

```
  size_t i, n, rank[N_MAX];
  static NagError fail;

  fail.print = TRUE;
  /* Skip heading in data file */
  Vscanf("%*[^\n]");
  Vprintf("m01dsc Example Program Results\n\n");
  Vscanf("%ld%ld", &n, &step);
  if (n>=0 && step!=0)
    {
      for (i=0; i<n; ++i)
        Vscanf("%lf", &vec[i]);
      m01dsc((Pointer) vec, n, step*(ptrdiff_t)(sizeof(double)), compare,
             Nag_Ascending, rank, &fail);
      if (fail.code != NE_NOERROR)
        exit(EXIT_FAILURE);
      Vprintf("    Data      Rank\n");
      for (i=0; i<n; ++i)
        Vprintf("   %7.4f   %4d\n", vec[i], rank[i]);
      exit(EXIT_SUCCESS);
    }
  else
    {
      Vfprintf(stderr, "Data error: program terminated.\n");
      exit(EXIT_FAILURE);
    }
}
```

8.2. Program Data

```
m01dsc Example Program Data
12
1
5.3 4.6 7.8 1.7 5.3 9.9 3.2 4.3 7.8 4.5 1.2 7.6
```

8.3. Program Results

```
m01dsc Example Program Results

    Data      Rank
    5.3000       6
    4.6000       5
    7.8000       9
    1.7000       1
    5.3000       7
    9.9000      11
    3.2000       2
    4.3000       3
    7.8000      10
    4.5000       4
    1.2000       0
    7.6000       8
```