nag_double_sort (m01cac)

1. Purpose

nag_double_sort (m01cac) rearranges a vector of real numbers into ascending or descending order.

2. Specification

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

void nag_double_sort(double vec[], size_t n, Nag_SortOrder order, NagError *fail)
```

3. Description

nag_double_sort is based on Singleton's implementation of the 'median-of-three' Quicksort algorithm, see Singleton (1969), but with two additional modifications. First, small subfiles are sorted by an insertion sort on a separate final pass, see 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[n]

Input: elements of **vec** must contain real values to be sorted.
Output: these values are rearranged into sorted order.

n

Input: the length of **vec**.
Constraint: $\mathbf{n} \geq 1$.

order

Input: Specifies whether the array will 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 1: $\mathbf{n} = \langle value \rangle$.

NE_INT_ARG_GT

On entry, **n** must not be greater than $\langle value \rangle$: $\mathbf{n} = \langle value \rangle$.
This parameter is limited by an implementation-dependent size which is printed in the error message.

NE_BAD_PARAM

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

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

None.

8. Example

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

8.1. Program Text

```
/* nag_double_sort(m01cac) Example Program
 *
 * Copyright 1990 Numerical Algorithms Group.
 *
 * Mark 1, 1990.
 */

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

#define NMAX 50

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

  /* Skip heading in data file */
  Vscanf("%*[^\n]");
  Vprintf("m01cac Example Program Results\n");
  Vscanf("%ld",&n);
  if (n<0 || n>NMAX)
    {
      Vfprintf(stderr, "n is out of range: n = %5ld\n", n);
      exit(EXIT_FAILURE);
    }
  for (i=0; i<n; ++i)
    Vscanf("%lf",&vec[i]);
  fail.print = TRUE;
  m01cac(vec, (size_t) n, Nag_Ascending, &fail);
  if (fail.code != NE_NOERROR)
    exit (EXIT_FAILURE);
  Vprintf("Sorted numbers\n\n");
  for (i=0; i<n; ++i)
    Vprintf("%10.6g%c",vec[i],(i%7==6 || i==n-1) ? '\n' : ' ');
  exit(EXIT_SUCCESS);
}
```

8.2. Program Data

```
m01cac Example Program Data
16
1.3 5.9 4.1 2.3 0.5 5.8 1.3 6.5
2.3 0.5 6.5 9.9 2.1 1.1 1.2 8.6
```

8.3. Program Results

```
m01cac Example Program Results
Sorted numbers

       0.5        0.5        1.1        1.2        1.3        1.3        2.1
       2.3        2.3        4.1        5.8        5.9        6.5        6.5
       8.6        9.9
```