NAG Library Function Document

nag_double_sort (m01cac)

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

2 Specification

```c
#include <nag.h>
#include <nagm01.h>
void nag_double_sort (double vec[], size_t n, Nag_SortOrder order, NagError *fail)
```

3 Description

nag_double_sort (m01cac) 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 References

Sedgewick R (1978) Implementing Quicksort programs Comm. ACM 21 847–857

5 Arguments

1: vec[n] – double  
   Input/Output  
   On entry: elements of vec must contain real values to be sorted.  
   On exit: these values are rearranged into sorted order.

2: n – size_t  
   Input  
   On entry: the length of vec.  
   Constraint: n ≥ 1.

3: order – Nag_SortOrder  
   Input  
   On entry: specifies whether the array will be sorted into ascending or descending order.  
   Constraint: order = Nag_Ascending or Nag_Descending.

4: fail – NagError *  
   Input/Output  
   The NAG error argument (see Section 3.6 in the Essential Introduction).
6 Error Indicators and Warnings

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

**NE_INT_ARG_GT**
On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \leq \langle \text{value} \rangle \). This argument is limited by an implementation-dependent size which is printed in the error message.

**NE_INT_ARG_LT**
On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 1 \).

7 Accuracy
Not applicable.

8 Parallelism and Performance
Not applicable.

9 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.

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

10.1 Program Text
/* nag_double_sort (m01cac) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 1, 1990. */
/* Mark 8 revised, 2004. */
*
#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nag_stddef.h>
#include <nagm01.h>

int main(void)
{
    Integer exit_status = 0, i, n;
    NagError fail;
    double *vec = 0;

    INIT_FAIL(fail);

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n]");
    #else
    scanf("%*[\n]");
    */
#ifdef _WIN32
scanf_s(%"NAG_IFMT"", &n);
#else
scanf(%"NAG_IFMT"", &n);
#endif
if (n >= 1)
{
    if (!(vec = NAG_ALLOC(n, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else
{
    printf("Invalid n.\n");
    exit_status = 1;
    return exit_status;
}
for (i = 0; i < n; ++i)
#ifdef _WIN32
    scanf_s(%"lf", &vec[i]);
#else
    scanf(%"lf", &vec[i]);
#endif
/* nag_double_sort (m01cac).
 * Quicksort of set of values of data type double
 */
nag_double_sort(vec, (size_t) n, Nag_Ascending, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_double_sort (m01cac).
%s
", fail.message);
    exit_status = 1;
    goto END;
}
printf("Sorted numbers\n");
for (i = 0; i < n; ++i)
    printf("%10.6g%c", vec[i], (i%7 == 6 || i == n-1)?'\n':' ');
END:
NAG_FREE(vec);
return exit_status;
}

10.2 Program Data
nag_double_sort (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

10.3 Program Results
nag_double_sort (m01cac) Example Program Results
Sorted numbers

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>1.1</td>
<td>1.2</td>
<td>1.3</td>
<td>1.3</td>
<td>2.1</td>
</tr>
<tr>
<td>2.3</td>
<td>2.3</td>
<td>4.1</td>
<td>5.8</td>
<td>5.9</td>
<td>6.5</td>
<td>6.5</td>
</tr>
<tr>
<td>8.6</td>
<td>9.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>