Contents

1 Test of Hash and Insertion ranking  1
   1.1 Main program .............................. 2
      1.1.1 Timing A Ranking Routine ................. 4

2 Formatting rules for HPF/F90 files  5
1. Test of Hash and Insertion ranking
1.1 Main program

This program performs a timing of the six major ranking algorithms provided in the module Sorting_Ranking for arrays of different sizes and different initial disorders.

"TimeRanking.f90" 1.1 =

PROGRAM Time_Ranking
USE Precision
USE Sorting_Ranking
USE Random_Numbers
USE Simple_Graphics
USE StopWatch
IMPLICIT NONE

INTEGER :: size_array, min_size, max_size, n_repeated
INTEGER, DIMENSION (:), ALLOCATABLE :: permutation
INTEGER, DIMENSION (:), ALLOCATABLE :: i_array
REAL, DIMENSION (:), ALLOCATABLE :: r_array

TYPE (WATCHTYPE) :: timer, timers
REAL :: disorder, elapsed_time, mean, std
INTEGER :: i, j, reps, n_reps, n_calls, n_steps
LOGICAL :: test
CHARACTER (LEN = 1) :: distribution, method
CHARACTER (LEN = 25) :: buffer, file_name
REAL, DIMENSION (; :, :), ALLOCATABLE :: timing_matrix

CALL CREATE_WATCH (timer)
CALL CREATE_WATCH (timers)
CALL UnpredictableSeeds()

depth = F

WRITE(*, *) "Enter min_size, max_size as LOG10 of actual array size: 
READ(*, *) min_size, max_size
WRITE(*, *) "Enter the number of disorders to time (> = 2): 
READ(*, *) n_steps
WRITE(*, *) "Distribution for random numbers in array (Uniform or Normal): 
READ(*, *) distribution
ALLOCATE (timing_matrix(6, n_steps, max_size - min_size + 1)) // The timing results for 6 sorts

OPEN (UNIT = 10, FILE = "TimeRanking.full.dat", STATUS = "REPLACE", ACTION = "WRITE")
OPEN (UNIT = 20, FILE = "TimeRanking.short.dat", STATUS = "REPLACE", ACTION = "WRITE")
OPEN (UNIT = 30, FILE = "TimeRanking.matrix.dat", STATUS = "REPLACE", ACTION = "WRITE")

ChangeSize: DO j = min_size, max_size
    size_array = 10^j
    ALLOCATE (permutation(size_array), r_array(size_array))

ChangeDisorder: DO i = 1, n_steps
    disorder = REAL(i - 1) / REAL(n_steps - 1)
    WRITE (*, *) "Now doing arrays of size: ", size_array, " disorder = ", disorder
    WRITE (*, *) "-----------------------------------"
    WRITE (10, *) "Now doing arrays of size: ", size_array, " disorder = ", disorder
    WRITE (10, *) "-----------------------------------"
    WRITE (20, FMT = "(2I3,F10.2,110)", ADVANCE = "NO") i, j, disorder, size_array

...
IF (distribution == "U") THEN
    CALL RandomUniform(r_array) // range = (/ 0.0, 1.0 /) by default
    mean = 0.5
    std = 0.5 / SQRT(3.0)
ELSE
    CALL RandomNormal(r_array) // mean_std = (/ 0.0, 1.0 /) by default
    mean = 0.0
    std = 1.0
END IF
CALL MergeRank(r_array, permutation) // Order then disorder the array
r_array = r_array(permutation)
CALL DisorderPermutation(permutation = permutation, disorder = disorder,
    disorder_distribution = 'N')
r_array = r_array(permutation) // Now disorder the sorted array
_TIMERank("ORDERPACK's MergeRank", _CALLMERGE, 1)
IF (LOG(REAL(size_array)) <= -0.65 * LOG(disorder + 0.01) + 9.90) THEN // Empirical test
    _TIMERank("InsertionRank", _CALLINSERTION, 2)
ELSE
    _TIMERank("DummyInsertionRank", _DUMMY, 2)
END IF
IF (disorder < 0.5) || ((disorder >= 0.5) && (size_array <= 1500000)) THEN
    _TIMERank("ShellRank", _CALLSHELL, 3)
ELSE
    _TIMERank("DummyShellRank", _DUMMY, 3)
END IF
_TIMERank("QuickRank", _CALLQUICK, 4)
_TIMERank("HashRank", _CALLHASH, 5)
_TIMERank("RadixRank", _CALLRADIX, 6)
WRITE(*, *) "--------------------------------------------------------------------------------"
WRITE(10, *) "-------------------------------------------------------------------------------"
WRITE(20, *) // A new line
CALL FLUSH(10); ;
CALL FLUSH(20) // Flush the I/O buffers
END DO ChangeDisorder
DEALLOCATE (r_array, permutation)
END DO ChangeSize
WRITE(30, *) SHAPE(timing_matrix)
WRITE(30, *) timing_matrix
CLOSE (10)
CLOSE (20)
CLOSE (30)
END PROGRAM Time_Ranking
1.1.1 Timing A Ranking Routine

"TimeRanking.t90" 1.1.1 ≡

```c
@m  _CALLMERGE
call MergeRank(r_array, permutation)
@m  _CALLINSERTION
call ShellInsertionRank(array = r_array, permutation = permutation, partially_ranked = F, method = "Insertion")
@m  _CALLSHELL
call ShellInsertionRank(array = r_array, permutation = permutation, partially_ranked = F, method = "Shell", disorder = disorder)
@m  _CALLQUICK
call QuickRank(array = r_array, permutation = permutation, mean_value = mean, standard_deviation = std, partially_ranked = F, pivot_selection = distribution)
@m  _CALLHASH
call HashRank(array = r_array, permutation = permutation, mean_value = mean, standard_deviation = std, partially_ranked = F, distribution = distribution)
@m  _CALLRADIX
call RadixRank(array = TRANSFER(r_array, MOLD = 1, SIZE = size_array), permutation = permutation, partially_ranked = F)
@m  _TIMERANK(Rank, _CallRanking, 1D)

 n_swaps = 0
 n_reps = 0
 n_calls = MAX(1, 10000 / size_array)  // To avoid timing overhead influence
 call RESET_WATCH(timer)
 do n_reps = n_reps + n_calls
    call START_WATCH(timer)
    do reps = 1, n_calls
       _CallRanking
    end do
    call STOP_WATCH(timer)
    call READ_WATCH(read_result = elapsed_time, watch = timer, clock = "cpu")
 if (elapsed_time > 3.0) exit  // Enough accuracy for now
 end do
timing_matrix(id, i, j) = elapsed_time / REAL(n_reps)
 WRITE(*, *) Rank, " took (s)", elapsed_time / REAL(n_reps)
 WRITE(*, *) "Number of swaps=", n_swaps / n_reps
 WRITE(10, *) Rank, " took (s)": elapsed_time / REAL(n_reps)
 WRITE(10, *) "Number of swaps=", n_swaps / n_reps
 WRITE(20, FMT = "(E10.3)", advance = "NO") elapsed_time / REAL(n_reps)
```

[HPF2Formatting.hweb]
2 Formatting rules for HPF/F90 files

These are just same auxiliary formatting rules and useful macros I use from time to time.

```f90
@m _GENERICINTERFACE(generic_name,...)
   INTERFACE generic_name
   MODULE PROCEDURE #.
   END INTERFACE generic_name
@m _DECLARE_TWORD(...)
   INTEGER :: #.
@m _DECLARE_TWP(...)
   INTEGER (KIND = i_wp) :: #.
@m _DECLARE_RWP(...)
   REAL (KIND = r_wp) :: #.
@m _DECLARE_RSP(...)
   REAL (KIND = r_sp) :: #.
@m _DECLARE_RDP(...)
   REAL (KIND = r_dp) :: #.
@m _FULLEXTENT(rank) : $DO (DIM, 2, rank) { , }$DO (DIM, 2, rank) { , }
@m _VARSEQUENCE(variable, start, end)
   variable #: start $DO (DIM, $EVAL (start + 1), end) { , variable & DIM }$DO (DIM, $EVAL (start + 1), end) { , variable & DIM }
@m _NESTEDLOOPSTART(variable, array, rank)
   $DO (DIM, rank, 1, -1) { DO variable & DIM = LBOUND(array, DIM), UBOUND(array, DIM) }$DO (DIM, rank, 1, -1) { DO variable & DIM = LBOUND(array, DIM), UBOUND(array, DIM) }
@m _NESTEDLOOPEnd(rank) $DO (DIM, 1, rank) { END DO }$DO (DIM, 1, rank) { END DO }
@m _DUMMY(...)
@m _DISPLAYARRAY(message, array)
   IF (SIZE(array) <= 20) THEN
      WRITE(*, "(a)") message
      WRITE(*, "(20G10.2)") array
   END IF
```