|  |  |
| --- | --- |
| order {base} | R Documentation |

**Ordering Permutation**

**Description**

order returns a permutation which rearranges its first argument into ascending or descending order, breaking ties by further arguments. sort.list is the same, using only one argument.

See the examples for how to use these functions to sort data frames, etc.

**Usage**

order(..., na.last = TRUE, decreasing = FALSE,

method = c("shell", "radix"))

sort.list(x, partial = NULL, na.last = TRUE, decreasing = FALSE,

method = c("shell", "quick", "radix"))

**Arguments**

|  |  |
| --- | --- |
| ... | a sequence of numeric, complex, character or logical vectors, all of the same length, or a classed **R** object. |
| x | an atomic vector. |
| partial | vector of indices for partial sorting. (Non-NULL values are not implemented.) |
| decreasing | logical. Should the sort order be increasing or decreasing? For the "radix" method, this can be a vector of length equal to the number of arguments in .... For the other methods, it must be length one. |
| na.last | for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed (see ‘Note’.) |
| method | the method to be used: partial matches are allowed. The default is "shell" except for some special cases: see ‘Details’. For details of methods "shell", "quick", and "radix", see the help for [sort](http://127.0.0.1:14695/library/base/help/sort). |

**Details**

In the case of ties in the first vector, values in the second are used to break the ties. If the values are still tied, values in the later arguments are used to break the tie (see the first example). The sort used is *stable* (except for method = "quick"), so any unresolved ties will be left in their original ordering.

Complex values are sorted first by the real part, then the imaginary part.

Except for method "radix", the sort order for character vectors will depend on the collating sequence of the locale in use: see [Comparison](http://127.0.0.1:14695/library/base/help/Comparison).

The "shell" method is generally the safest bet and is the default method, except for factors and integer vectors, where "radix" is assumed. Method "radix" stably sorts logical, numeric and character vectors in linear time. It outperforms the other methods, although there are caveats (see [sort](http://127.0.0.1:14695/library/base/help/sort)). Method "quick" for sort.list is only supported for numeric x with na.last = NA, is not stable, and is slower than "radix". The "radix" method has less precision when sorting real-valued numbers.

partial = NULL is supported for compatibility with other implementations of S, but no other values are accepted and ordering is always complete.

For a classed **R** object, the sort order is taken from [xtfrm](http://127.0.0.1:14695/library/base/help/xtfrm): as its help page notes, this can be slow unless a suitable method has been defined or [is.numeric](http://127.0.0.1:14695/library/base/help/is.numeric)(x) is true. For factors, this sorts on the internal codes, which is particularly appropriate for ordered factors.

**Value**

An integer vector unless any of the inputs has *2^31* or more elements, when it is a double vector.

**Warning**

In programmatic use it is unsafe to name the ... arguments, as the names could match current or future control arguments such as decreasing. A sometimes-encountered unsafe practice is to call do.call('order', df\_obj) where df\_obj might be a data frame: copy df\_obj and remove any names, for example using [unname](http://127.0.0.1:14695/library/base/help/unname).

**Note**

sort.list can get called by mistake as a method for [sort](http://127.0.0.1:14695/library/base/help/sort) with a list argument: it gives a suitable error message for list x.

There is a historical difference in behaviour for na.last = NA: sort.list removes the NAs and then computes the order amongst the remaining elements: order computes the order amongst the non-NA elements of the original vector. Thus

x[order(x, na.last = NA)]

zz <- x[!is.na(x)]; zz[sort.list(x, na.last = NA)]

both sort the non-NA values of x.

Prior to **R** 3.1.0 method = "radix" was only supported for non-negative integers. Prior to **R** 3.3.0 it was only supported for integers of range less than 100,000.

**References**

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Knuth, D. E. (1998) *The Art of Computer Programming, Volume 3: Sorting and Searching.* 2nd ed. Addison-Wesley.

**See Also**

[sort](http://127.0.0.1:14695/library/base/help/sort), [rank](http://127.0.0.1:14695/library/base/help/rank), [xtfrm](http://127.0.0.1:14695/library/base/help/xtfrm).

**Examples**

require(stats)

(ii <- order(x <- c(1,1,3:1,1:4,3), y <- c(9,9:1), z <- c(2,1:9)))

## 6 5 2 1 7 4 10 8 3 9

rbind(x, y, z)[,ii] # shows the reordering (ties via 2nd & 3rd arg)

## Suppose we wanted descending order on y.

## A simple solution for numeric 'y' is

rbind(x, y, z)[, order(x, -y, z)]

## More generally we can make use of xtfrm

cy <- as.character(y)

rbind(x, y, z)[, order(x, -xtfrm(cy), z)]

## The radix sort supports multiple 'decreasing' values:

rbind(x, y, z)[, order(x, cy, z, decreasing = c(FALSE, TRUE, FALSE),

method="radix")]

## Sorting data frames:

dd <- transform(data.frame(x, y, z),

z = factor(z, labels = LETTERS[9:1]))

## Either as above {for factor 'z' : using internal coding}:

dd[ order(x, -y, z), ]

## or along 1st column, ties along 2nd, ... \*arbitrary\* no.{columns}:

dd[ do.call(order, dd), ]

set.seed(1) # reproducible example:

d4 <- data.frame(x = round( rnorm(100)), y = round(10\*runif(100)),

z = round( 8\*rnorm(100)), u = round(50\*runif(100)))

(d4s <- d4[ do.call(order, d4), ])

(i <- which(diff(d4s[, 3]) == 0))

# in 2 places, needed 3 cols to break ties:

d4s[ rbind(i, i+1), ]

## rearrange matched vectors so that the first is in ascending order

x <- c(5:1, 6:8, 12:9)

y <- (x - 5)^2

o <- order(x)

rbind(x[o], y[o])

## tests of na.last

a <- c(4, 3, 2, NA, 1)

b <- c(4, NA, 2, 7, 1)

z <- cbind(a, b)

(o <- order(a, b)); z[o, ]

(o <- order(a, b, na.last = FALSE)); z[o, ]

(o <- order(a, b, na.last = NA)); z[o, ]

## speed examples on an average laptop for long vectors:

## factor/small-valued integers:

x <- factor(sample(letters, 1e7, replace = TRUE))

system.time(o <- sort.list(x, method = "quick", na.last = NA)) # 0.1 sec

stopifnot(!is.unsorted(x[o]))

system.time(o <- sort.list(x, method = "radix")) # 0.05 sec, 2X faster

stopifnot(!is.unsorted(x[o]))

## large-valued integers:

xx <- sample(1:200000, 1e7, replace = TRUE)

system.time(o <- sort.list(xx, method = "quick", na.last = NA)) # 0.3 sec

system.time(o <- sort.list(xx, method = "radix")) # 0.2 sec

## character vectors:

xx <- sample(state.name, 1e6, replace = TRUE)

system.time(o <- sort.list(xx, method = "shell")) # 2 sec

system.time(o <- sort.list(xx, method = "radix")) # 0.007 sec, 300X faster

## double vectors:

xx <- rnorm(1e6)

system.time(o <- sort.list(xx, method = "shell")) # 0.4 sec

system.time(o <- sort.list(xx, method = "quick", na.last = NA)) # 0.1 sec

system.time(o <- sort.list(xx, method = "radix")) # 0.05 sec, 2X faster