

# Package ‘disordR’

January 13, 2024

**Type** Package

**Title** Non-Ordered Vectors

**Version** 0.9-8.2

**Depends** methods,Matrix (>= 1.3-3)

**Imports** digest

**Suggests** mvp,knitr,rmarkdown,testthat

**VignetteBuilder** knitr

**Maintainer** Robin K. S. Hankin <hankin.robin@gmail.com>

**Description** Functionality for manipulating values of associative maps. The package is designed to be used with the 'mvp' class of packages that use the STL map class: its purpose is to trap plausible idiom that is ill-defined (implementation-specific) and return an informative error, rather than returning a possibly incorrect result. To cite the package in publications please use Hankin (2022) <doi:10.48550/ARXIV.2210.03856>.

**License** GPL (>= 2)

**URL** <https://github.com/RobinHankin/disordR>

**BugReports** <https://github.com/RobinHankin/disordR/issues>

**NeedsCompilation** no

**Author** Robin K. S. Hankin [aut, cre] (<<https://orcid.org/0000-0001-5982-0415>>)

**Repository** CRAN

**Date/Publication** 2024-01-13 19:10:05 UTC

## R topics documented:

Arith . . . . .	2
c . . . . .	3
Compare-methods . . . . .	4
consistent . . . . .	4
disindex-class . . . . .	5

disord . . . . .	6
disord-class . . . . .	8
drop . . . . .	8
extract . . . . .	10
Logic . . . . .	12
misc . . . . .	13
rdis . . . . .	15
show . . . . .	15
summary.disordR . . . . .	16

<b>Index</b>	<b>18</b>
--------------	-----------

---

Arith	<i>Arithmetic operations</i>
-------	------------------------------

---

### Description

Arithmetic operations including low-level helper functions

### Usage

```

disord_inverse(a)
disord_mod_disord(a,b)
disord_mod_numeric(a,b)
disord_negative(a)
disord_plus_disord(a,b)
disord_plus_numeric(a,b)
disord_power_disord(a,b)
disord_power_numeric(a,b)
numeric_power_disord(a,b)
disord_prod_disord(a,b)
disord_prod_numeric(a,b)
disord_arith_unary(e1,e2)
disord_arith_disord(e1,e2)
disord_arith_numeric(e1,e2)
numeric_arith_disord(e1,e2)

```

### Arguments

a,b	at least one is a disord object
e1,e2	Formal arguments for S4 dispatch

### Details

Basic low-level arithmetic operations, intended to be called from S4 dispatch.

These functions return a disord object or a regular vector as appropriate. Consistency is required. The hash is set to be that of the disord object if appropriate.

**Value**

Return a disord object or logical

**Methods**

**Arith** signature(e1="disord", e2="disord"): Dispatched to disord\_arith\_disord()

**Arith** signature(e1="disord", e2="numeric"): Dispatched to disord\_arith\_numeric()

**Arith** signature(e1="numeric", e2="disord"): Dispatched to numeric\_arith\_disord()

**Arith** signature(e1="disord", e2="missing"): Dispatched to disord\_arith\_unary()

**Author(s)**

Robin K. S. Hankin

**Examples**

```
a <- rdis()
a
a + 2*a
a > 5
a[a > 5] <- a[a > 5] + 100
a
```

---

c

*Concatenation*

---

**Description**

Concatenation simply does not make sense for disord objects.

**Value**

Returns an error.

**Note**

I could not figure out how to stop idiom like “c(1, rdis())” from returning a result. Just don’t use it, OK?

**Author(s)**

Robin K. S. Hankin

---

Compare-methods	<i>Methods for comparison of disord objects</i>
-----------------	---

---

### Description

Arithmetic comparison methods (greater than, etc) for disord objects.

### Methods

**Compare** signature(e1="disord", e2="disord"): Dispatched to disord\_compare\_disord()

**Compare** signature(e1="disord", e2="ANY"): Dispatched to disord\_compare\_any()

**Compare** signature(e1="ANY", e2="disord"): Dispatched to any\_compare\_disord()

### Note

All the comparison methods use drop=TRUE to avoid inconsistent results when all the values are the same [that is, all TRUE or all FALSE]. Comparing two disord objects requires their hash code to agree as per **disordR** discipline. Comparing a disord with a numeric returns a disord object. In each case, the hash code of the original object is preserved in the returned value.

### Examples

```
rdis() > 4
rdis() > 1000
```

---

consistent	<i>Check for consistency</i>
------------	------------------------------

---

### Description

The **disordR** package is designed to make permitted operations transparent and to prevent forbidden operations from being executed.

Function `consistent()` checks for matching hash codes of its arguments and returns a Boolean. It is called by function `check_matching_hash()` which either returns TRUE or reports an informative error message if not.

### Usage

```
consistent(x,y)
x %~% y
check_matching_hash(e1,e2,use=NULL)
```

**Arguments**

`x, y, e1, e2`      Objects of class `disord`

`use`              optional object designed to give a more intelligible error message; typically `match.call()`

**Details**

Function `consistent()` checks that its arguments have the same hash code, and thus their elements can be paired up (e.g. added). Idiom `a %% b` is equivalent to `consistent(a, b)`.

The package generally checks for consistency with function `check_matching_hash()` which provides some helpful diagnostics if `consistent()` finds a hash mismatch.

**Value**

Boolean or an error as appropriate

**Author(s)**

Robin K. S. Hankin

**See Also**

[disord](#)

**Examples**

```
# rdis() + rdis() # this would make check_matching_hash() report an error, if executed
```

---

disindex-class      *Experimental class "disindex"*

---

**Description**

Experimental `disindex` class provides a `disordR`-compliant method for indexing `disord` objects. The idea is that `which(x)`, where `x` is Boolean of class `disord`, should have meaning under `disordR` discipline. Thus `which()` gives a `disindex` object. This object can be used as an index for other `disord` objects. One application would be the `dismat` class of matrices, currently under development.

Function `values()` coerces its argument to an integer vector.

**Objects from the Class**

Objects can be created by calls of the form `new("disindex", ...)`, although `which()` is more natural.

**Slots**

value: Numeric vector

hash: Object of class character that specifies the hash code

**Author(s)**

Robin K. S. Hankin

**Examples**

```
(x <- disord(c(1,2,1,2,2,7)))  
  
x==2  
w <- which(x==2)  
w  
  
x[w] <- 100  
x
```

---

disord

*Functionality for disord objects*

---

**Description**

Allows arithmetic operators to be used for disord objects; the canonical application is coefficients of multivariate polynomials (as in the **mvp** package). The issue is that the storage order of disord objects is implementation-specific but the order (whatever it is) must be consistent between the list of keys and values in an associative array.

**Usage**

```
is.disord(x)  
hash(x)  
hashcal(x,ultra_strict=FALSE)  
disord(v,h,drop=TRUE)  
elements(x)
```

**Arguments**

x	Object of class disord
v	Vector of coefficients
h	Hash code
drop	Boolean, with default FALSE meaning to return a disord object and TRUE meaning to call drop() before returning
ultra_strict	Boolean, with default FALSE meaning to use just x to generate the hash, and TRUE meaning to use the date and a random number as well [this ensures that the hash is generated only once]

**Details**

A detailed vignette is provided that motivates the package. In applications such as the **mvp** or **cliford** packages, the user will not need to even think about the **disordR** package: it works in the background. The purpose of the package is to trap plausible idiom that is ill-defined (implementation-specific) and return an informative error, rather than returning a possibly incorrect result.

The package provides a single S4 class, `disord`, which has two slots, `.Data` and `hash`.

Function `disord()` takes an R object such as a vector or list and returns a `disord` object, which is useful in the context of the STL map class.

Function `hash()` returns the hash of an object (compare `hashcal()` which is used to actually calculate the hash code).

The package detects acceptable and forbidden operations using hash codes: function `consistent()` checks for its arguments having the same hash code, and thus their elements can be paired up (e.g. added). Idiomatically, `a %~% b` is equivalent to `consistent(a, b)`.

Function `elements()` takes a `disord` and returns a regular R object, typically a vector or a list.

**Value**

Boolean, hash code, or object of class `disord` as appropriate.

**Author(s)**

Robin K. S. Hankin

**Examples**

```
(a <- rdis())
(b <- rdis())

a + 2*a + 2^a # fine
# a + b # this would give an error if executed

a[a<0.5] <- 0 # round down; replacement works as expected

elements(a)
```

---

disord-class	<i>Class "disord"</i>
--------------	-----------------------

---

**Description**

The `disord` class provides basic arithmetic and `extract/replace` methods for `disord` objects.

**Objects from the Class**

Objects can be created by calls of the form `new("disord", ...)`, although functions `disord()` and (eventually) as `.disord()` are more user-friendly.

**Slots**

`.Data`: Object of class `vector` that specifies the elements

`hash`: Object of class `character` that specifies the hash code

**Author(s)**

Robin K. S. Hankin

**Examples**

```
showClass("disord")
```

---

drop	<i>Drop redundant information</i>
------	-----------------------------------

---

**Description**

Coerce `disord` objects to `vector` when this makes sense

**Usage**

```
drop(x)  
allsame(x)
```

**Arguments**

`x` `disord` object



**Details**

If one has a `disord` object all of whose elements are identical, one usually wants to drop the `disord` attribute and coerce to a vector. This can be done without breaking `disordR` discipline. Function `disord()` takes a `drop` argument, defaulting to `TRUE`, which drops the `disord` class from its return value if all the elements are the same.

Similarly, function `drop()` takes a `disord` object and if all elements are identical it returns the elements in the form of a vector. Some extraction methods take a `drop` argument, which does the same thing if `TRUE`. This is only useful for `disord` objects created with `disord(...,drop=FALSE)`

The `drop` functionality is conceptually similar to the `drop` argument of base `R`'s array extraction, as in

```
a <- matrix(1:30,5,6)
a[1,,drop=TRUE]
a[1,,drop=FALSE]
```

Function `allsame()` takes a vector and returns `TRUE` if all elements are identical.

**Value**

Function `drop()` returns either a vector or object of class `disord` as appropriate; `allsame()` returns a Boolean.

**Author(s)**

Robin K. S. Hankin

**Examples**

```
disord(c(3,3,3,3,3))           # default is drop=TRUE
disord(c(3,3,3,3,3),drop=FALSE) # retains disord class

drop(disord(c(3,3,3,3),drop=FALSE))

## In extraction, argument drop discards disorderliness when possible:
a <- rdis()
a
a[] <- 6 # a becomes a vector
a
```

---

 extract

*Extraction and replacement methods for class "disord"*


---

## Description

The `disord` class provides basic arithmetic and `extract/replace` methods for `disord` objects.

Class `index` is taken from the excellent **Matrix** package and is a `setClassUnion()` of classes `numeric`, `logical`, and `character`.

## Methods

```
[ signature(x = "disord", i = "ANY", j = "ANY"): ...
[ signature(x = "disord", i = "index", j = "index"): ...
[ signature(x = "disord", i = "index", j = "missing"): ...
[ signature(x = "disord", i = "missing", j = "index"): ...
[ signature(x = "disord", i = "missing", j = "missing"): ...
[ signature(x = "disord", i = "matrix", j = "missing"): ...
[<- signature(x = "disord", i = "index", j = "index"): ...
[<- signature(x = "disord", i = "index", j = "missing"): ...
[<- signature(x = "disord", i = "missing", j = "index"): ...
[<- signature(x = "disord", i = "matrix", j = "missing"): ...
[<- signature(x = "disord", i = "missing", j = "missing"): ...
[[ signature(x = "disord", i = "index"): ...
[[<- signature(x = "disord", i = "index", value="ANY"): ...
[ signature(x="disord",i="disindex",j="missing",drop="ANY"): ...
[ signature(x="disord",i="disindex",j="ANY",drop="ANY"): ...
[ signature(x="ANY",i="disindex",j="ANY",drop="ANY"): ...
[ signature(x="disord",i="disindex",j="missing",value="ANY"): ...
[ signature(x="disord",i="disindex",j="ANY",value="ANY"): ...
[<- signature(x="disord",i="disindex",j="missing",drop="ANY"): ...
[[ signature("disord",i="disindex"): ...
[[ signature("ANY",i="disindex"): ...
[[<- signature(x="disord",i="disindex",j="missing",value="ANY") ...
[[<- signature(x="ANY",i="disindex",j="ANY",value="ANY") ...
```

The extraction method takes a `drop` argument which if `TRUE`, returns the `drop()` of its value. Extraction, as in `x[i]`, is rarely useful. It is only defined if one extracts either all, or none, of the elements: anything else is undefined. Note that the hash code is unchanged if all elements are extracted (because the order might have changed) but unchanged if none are (because there is only one way to extract no elements).

Missing arguments for extraction and replacement are slightly idiosyncratic. Extraction idiom such as `x[]` returns an object identical to `x` except for the hash code, which is changed. I can't quite see a sensible use-case for this, but the method allows one to define an object `y <- x[]` for which `x` and `y` are incompatible. Replacement idiom `x[] <- v` always coerces to a vector.

Double square extraction, as in `x[[i]]` and `x[[i]] <- value`, is via (experimental) `disindex` functionality.

### Note

Package versions prior to `disordR_0.0-9-6` allowed idiom such as

```
a <- disord(1:9)
a[a<3] + a[a>7]
```

but this is now disallowed. The issue is discussed in `inst/note_on_extraction.Rmd`.

### Author(s)

Robin K. S. Hankin

### See Also

[drop,misc](#)

### Examples

```
a <- disord(sample(9))
a
a + 6*a^2
a[a>5] # "give me all elements of a that exceed 5"

a[] # a disord object, same elements as 'a', but with a different hash

a[a<5] <- a[a<5] + 100 # "replace all elements of 'a' less than 5 with their value plus 100"
a

## Following expressions would return an error if executed:
if(FALSE){
  a[1]
  a[1] <- 44
  a[1:2] <- a[3:4]
}

b <- disord(sample(9))
## Following expressions would also return an error if executed:
if(FALSE){
  a+b # (not really an example of extraction)
  a[b>5]
  a[b>5] <- 100
  a[b>5] <- a[b>5] + 44
```

}

---

 Logic

 Logical operations
 

---

**Description**

Logical operations including low-level helper functions

**Usage**

```
disord_logical_negate(x)
disord_logic_disord(e1, e2)
disord_logic_any(e1, e2)
any_logic_disord(e1, e2)
```

**Arguments**

e1, e2, x      Formal arguments for S4 dispatch: logical disord object

**Details**

Basic low-level logical operations, intended to be called from S4 dispatch.

These functions return a logical disord object. appropriate. Consistency is required. The hash is set to be that of the disord object if appropriate.

**Value**

Return a disord object or logical

**Methods**

**Logic** signature(e1="disord", e2="disord"): Dispatched to disord\_logic\_disord()

**Logic** signature(e1="disord", e2="ANY"): Dispatched to disord\_logic\_any()

**Logic** signature(e1="ANY", e2="disord"): Dispatched to any\_logic\_disord()

**Author(s)**

Robin K. S. Hankin

**Examples**

```
a <- disord(1:7)
l <- a>3
sum(l)
any(l)
all(l | !l)
```

misc

*Miscellaneous functions***Description**

This page documents various functions that work for disords, and I will add to these from time to time as I add new functions that make sense for disord objects. Functions like `sin()` and `abs()` work as expected: they take and return disord objects with the same hash as `x` (which means that idiom like `x + sin(x)` is accepted). However, there are a few functions that are a little more involved:

- `rev()` reverses its argument and returns a disord object with a reversed hash, which ensures that `rev(rev(x))==x` (and the two are consistent).
- `sort()` returns a vector of sorted elements (not a disord)
- `length()` returns the length of the data component of the object
- `sapply(X, f)` returns a disord object which is the result of applying `f()` to each element of `X`.
- `match(x, table)` should behave as expected but note that if `table` is a disord, the result is not defined (because it is not known where the elements of `x` occur in `table`). Nevertheless `x %in% table` is defined and returns a disord object.
- `lapply(x, f)` returns `disord(lapply(elements(x), f, ...), h=hash(x))`. Note that double square bracket extraction, as in `x[[i]]`, is disallowed (see `extract.Rd`).
- `which()` returns a `disind` object when given a Boolean disord
- `unlist()` takes a disord list, flattens it and returns a disord vector. It requires the recursive flag of `base::unlist()` to be `TRUE`, which it is by default, interpreting this to mean “kill all the structure in any sublists”. If the list comprises only length-one vectors, the returned value retains the same hash as the argument; if not, a new hash is generated.
- `diff()` is undefined for disord objects.

**Arguments**

`x`                      Object of class `disord`

**Value**

Returns a `disord`

**Note**

Some functionality is not yet implemented. Factors, lists, and named vectors do not behave entirely consistently in the package; `paste()` gives inconsistent results when called with disords.

Also, `for()` loops are incompatible with disord discipline, as they impose an ordering (`for()` accesses the `.Data` slot of its argument, which is a regular R vector). Thus:

```

> (a <- disord(1:3))
A disord object with hash 555f6bea49e58a2c2541060a21c2d4f9078c3086 and elements
[1] 1 2 3
(in some order)
> for(i in a){print(i)}
[1] 1
[1] 2
[1] 3
>

```

Above, we see that `for()` uses the ordering of the `.Data` slot of S4 object `a`, even though `elements()` has not been explicitly called.

### Author(s)

Robin K. S. Hankin

### See Also

[extract](#)

### Examples

```

a <- disord(c(a=1,b=2,c=7))
a
names(a)
length(a)
sqrt(a)

# powers() and vars() in the mvp package return lists; see the vignette
# for more discussion.

l <- disord(list(3,6:9,1:10))
sapply(l,length)

unlist(l)

## Quick illustration of rev():

revstring <- function(s){paste(rev(unlist(strsplit(s, NULL))),collapse="")}
x <- rdis()
revstring(hash(x)) == hash(rev(x))

```

---

rdis	<i>Random disord objects</i>
------	------------------------------

---

**Description**

Returns a random disord object

**Usage**

```
rdis(n=9)
```

**Arguments**

n                   Set to sample from, as interpreted by `sample()`

**Details**

A simple disord object, intended as a quick “get you going” example

**Value**

A disord object.

**Author(s)**

Robin K. S. Hankin

**Examples**

```
rdis()  
rdis(99)  
rdis(letters)
```

---

show	<i>Print method for disord objects</i>
------	--

---

**Description**

Show methods for disorders

**Usage**

```
## S4 method for signature 'disord'  
show(x)  
disord_show(x)
```

**Arguments**

x                    Object of class disord

**Details**

The print method simply prints the object's hash and its elements, together with a reminder that the elements are listed in an implementation-specific order. Function `disord_show()` is a helper function, not really intended for the end-user.

**Author(s)**

Robin K. S. Hankin

**Examples**

```
print(rdis())
```

---

summary.disordR            *Summaries of disord objects*

---

**Description**

A summary method for disord objects, and a print method for summaries.

**Usage**

```
## S4 method for signature 'disord'
summary(object, ...)
## S4 method for signature 'disindex'
summary(object, ...)
## S3 method for class 'summary.disord'
print(x, ...)
```

**Arguments**

object, x            Object of class disord  
 ...                  Further arguments, currently ignored

**Details**

A `summary.disord` object is summary of a `disord` object `x`: a list with first element being the `hash(x)` and the second being `summary(elements(x))`. The print method is just a wrapper for this.

**Author(s)**

Robin K. S. Hankin



*summary.disordR*

17

### **Examples**

```
summary(rdis(1000))
```

# Index

!,disord-method (misc), 13

\* **classes**

- disindex-class, 5
- disord-class, 8

\* **math**

- Compare-methods, 4

\* **methods**

- Compare-methods, 4

\* **symbolmath**

- consistent, 4
- disord, 6

[ (extract), 10

[,ANY,disindex,ANY,ANY-method (extract), 10

[,ANY,disord,ANY-method (extract), 10

[,disord,ANY,ANY-method (extract), 10

[,disord,disindex,ANY,ANY-method (extract), 10

[,disord,disindex,missing,ANY-method (extract), 10

[,disord,disord,missing,ANY-method (extract), 10

[,disord,disord,missing-method (extract), 10

[,disord,index,ANY,ANY-method (extract), 10

[,disord,index,ANY-method (extract), 10

[,disord,index,index-method (extract), 10

[,disord,index,missing,ANY-method (extract), 10

[,disord,index,missing-method (extract), 10

[,disord,missing,index-method (extract), 10

[,disord,missing,missing,ANY-method (extract), 10

[,disord,missing,missing-method (extract), 10

[,disord-method (extract), 10

[.disord (extract), 10

[<- (extract), 10

[<-,disord,ANY,ANY-method (extract), 10

[<-,disord,disindex,ANY,ANY-method (extract), 10

[<-,disord,disindex,missing,ANY-method (extract), 10

[<-,disord,disord,missing,disord-method (extract), 10

[<-,disord,disord,missing-method (extract), 10

[<-,disord,index,ANY,ANY-method (extract), 10

[<-,disord,index,index-method (extract), 10

[<-,disord,index,missing,ANY-method (extract), 10

[<-,disord,index,missing,disord-method (extract), 10

[<-,disord,index,missing,numeric-method (extract), 10

[<-,disord,index,missing-method (extract), 10

[<-,disord,missing,index-method (extract), 10

[<-,disord,missing,missing,ANY-method (extract), 10

[<-,disord,missing,missing,disord-method (extract), 10

[<-,disord,missing,missing,numeric-method (extract), 10

[<-,disord,missing,missing-method (extract), 10

[<-,disord-method (extract), 10

[<-.disord (extract), 10

[[ (extract), 10

- [[, ANY, disindex-method (extract), 10
- [[, disord, disindex-method (extract), 10
- [[, disord, index-method (extract), 10
- [[<- , ANY, disindex, ANY, ANY-method (extract), 10
- [[<- , ANY, disindex, ANY-method (extract), 10
- [[<- , disord, disindex, ANY-method (extract), 10
- [[<- , disord, disindex, missing, ANY-method (extract), 10
- [[<- , disord, disindex, missing-method (extract), 10
- [[<- , disord, index, ANY-method (extract), 10
- [[<- , disord, index-method (extract), 10
- %~% (consistent), 4
- %in% (misc), 13
- %in%, ANY, disord-method (misc), 13
- %in%, disord, ANY-method (misc), 13
- %in%, disord, disord-method (misc), 13
- %in%, disord-method (misc), 13
- accessors (disord), 6
- allsame (drop), 8
- any\_compare\_disord (Compare-methods), 4
- any\_logic\_disord (Logic), 12
- Arith, 2
- Arith, ANY, disord-method (extract), 10
- Arith, disord, ANY-method (extract), 10
- Arith, disord, disord-method (extract), 10
- Arith, disord, missing-method (extract), 10
- as.character, disord-method (misc), 13
- as.complex, disord-method (misc), 13
- as.double, disord-method (misc), 13
- as.list, disord-method (misc), 13
- as.logical, disord-method (misc), 13
- as.numeric, disord-method (misc), 13
- as\_disord (disord), 6
- c, 3
- c, disord-method (c), 3
- c.disord (c), 3
- check\_matching\_hash (consistent), 4
- Compare, ANY, disord-method (Compare-methods), 4
- Compare, disord, ANY-method (Compare-methods), 4
- Compare, disord, disord-method (Compare-methods), 4
- Compare-methods, 4
- consistent, 4
- diff (misc), 13
- disindex (disindex-class), 5
- disindex-class, 5
- disindex\_show (Arith), 2
- disord, 5, 6
- disord-class, 8
- disord<- (disord), 6
- disord\_arith\_disord (Arith), 2
- disord\_arith\_numeric (Arith), 2
- disord\_arith\_unary (Arith), 2
- disord\_compare\_any (Compare-methods), 4
- disord\_compare\_disord (Compare-methods), 4
- disord\_inverse (Arith), 2
- disord\_logic (Logic), 12
- disord\_logic\_any (Logic), 12
- disord\_logic\_disord (Logic), 12
- disord\_logic\_missing (Logic), 12
- disord\_logic\_unary (Logic), 12
- disord\_logical\_negate (Logic), 12
- disord\_mod\_disord (Arith), 2
- disord\_mod\_numeric (Arith), 2
- disord\_negative (Arith), 2
- disord\_plus\_disord (Arith), 2
- disord\_plus\_numeric (Arith), 2
- disord\_positive (Arith), 2
- disord\_power\_disord (Arith), 2
- disord\_power\_numeric (Arith), 2
- disord\_prod\_disord (Arith), 2
- disord\_prod\_numeric (Arith), 2
- disord\_show (show), 15
- disord\_unary (Arith), 2
- drop, 8, 11
- drop, disord-method (drop), 8
- elements (disord), 6
- extract, 10, 14
- hash (disord), 6
- hashcal (disord), 6
- index-class (extract), 10
- is.consistent (consistent), 4
- is.disord (disord), 6

is.na (misc), 13  
is.na, disord-method (misc), 13  
is.na.disord (misc), 13  
is.na<- (misc), 13  
is.na<-, disord-method (misc), 13  
is.na<- .disord (misc), 13

lapply (misc), 13  
lapply, disord-method (misc), 13  
lapply.disord (misc), 13  
length (misc), 13  
length, disindex-method (misc), 13  
length, disord-method (misc), 13  
length.disindex (misc), 13  
length.disord (misc), 13  
length<- (misc), 13  
length<-, disord-method (misc), 13  
length<- .disord (misc), 13  
Logic, 12

match (misc), 13  
match, ANY, disord-method (misc), 13  
match, disord, ANY-method (misc), 13  
match, disord, disord-method (misc), 13  
match, disord-method (misc), 13  
misc, 11, 13

numeric\_arith\_disord (Arith), 2  
numeric\_mod\_disord (Arith), 2  
numeric\_power\_disord (Arith), 2

print (show), 15  
print, disord-method (show), 15  
print.disord (show), 15  
print.summary.disord (summary.disordR),  
16

rdis, 15  
rdisord (rdis), 15  
rdisordR (rdis), 15  
rev (misc), 13  
rev, disord-method (misc), 13  
rev.disord (misc), 13

sapply (misc), 13  
sapply, disord-method (misc), 13  
sapply.disord (misc), 13  
show, 15  
show, disord-method (show), 15  
sort (misc), 13  
sort, disord-method (misc), 13  
sort.disord (misc), 13  
summary, disindex-method  
(summary.disordR), 16  
summary, disord-method  
(summary.disordR), 16  
summary.disord (summary.disordR), 16  
summary.disordR, 16

unlist (misc), 13  
unlist, disord-method (misc), 13

values (disindex-class), 5

which, disindex-method (misc), 13  
which, disord-method (misc), 13