

Package ‘atime’

September 17, 2026

Type Package

Title Asymptotic Timing

Version 2026.9.17

Depends R ($\geq$ 3.5.0)

Description Computing and visualizing comparative asymptotic timings of different algorithms and code versions. Also includes functionality for comparing empirical timings with expected references such as linear or quadratic, [<https://en.wikipedia.org/wiki/Asymptotic_computational_complexity>](https://en.wikipedia.org/wiki/Asymptotic_computational_complexity) Also includes functionality for measuring asymptotic memory and other quantities.

License GPL-3

URL <https://github.com/tdhock/atime>, <https://atime-docs.netlify.app>

BugReports <https://github.com/tdhock/atime/issues>

Imports data.table, bench, lattice, gert, utils, stats, grDevices

SystemRequirements git command line program

Suggests directlabels, ggplot2, testthat, stringi, re2, binsegRcpp, grates, fastymd, wbs, changepoint, LOPART, PeakSegDisk, callr, readr, dplyr, tidyr, nc, RColorBrewer, Matrix, litedown

VignetteBuilder litedown

NeedsCompilation no

Author Toby Hocking [aut, cre] (ORCID: [<https://orcid.org/0000-0002-3146-0865>](https://orcid.org/0000-0002-3146-0865))

Maintainer Toby Hocking <toby.hocking@r-project.org>

Repository CRAN

Date/Publication 2026-09-17 20:50:21 UTC

Contents

time	2
time_grid	4
time_pkg	5
time_test	7
time_test_list	9
time_versions	10
time_versions_exprs	12
time_versions_remove	14
glob_find_replace	15
issue107	16
issue109	16
references_best	17
R_CMD_INSTALL	18
Index	19

time	<i>Asymptotic timing</i>
------	--------------------------

Description

Computation time and memory for several R expressions of several different data sizes.

Usage

```
time(  
  N = default_N(),  
  setup,  
  expr.list=NULL,  
  times=10,  
  seconds.limit=0.01,  
  verbose=FALSE,  
  result=NULL,  
  N.env.parent=NULL,  
  check=FALSE,  
  ...)
```

Arguments

- | | |
|---------------|--|
| N | numeric vector of at least two unique data sizes, default is 2^seq(2, 20). |
| setup | expression to evaluate for every data size, before timings. |
| expr.list | named list of expressions to time. |
| times | number of times to evaluate each timed expression. |
| seconds.limit | if the median timing of any expression exceeds this many seconds, then no timings for larger N are computed. |

verbose	logical, print messages after every data size?
result	logical: save the result of evaluating each expression? Or a function to compute a result, given the value obtained after evaluating each expression. Or NULL (default), meaning to only save if the result is a data frame with one row. If evaluation result is a data frame with one row, then the numeric column names will be saved as more units to analyze (in addition to kilobytes and seconds).
N.env.parent	environment to use as parent of environment created for each data size N, or NULL to use default parent env.
check	logical, error if results are not all equal? (default FALSE) Passed to mark .
...	named expressions to time.

Details

Each iteration involves first computing the setup expression, and then computing several times the ...expressions. For convenience, expressions may be specified either via code (...) or data (expr.list arg).

Value

list of class atime with elements unit.col.vec (character vector of column names to analyze), seconds.limit (numeric input param), measurements (data table of results).

Author(s)

Toby Dylan Hocking

See Also

[atime_grid](#) for avoiding repetition when measuring asymptotic properties of several similar expressions.

Examples

```
data.table::setDTthreads(1) # for CRAN.

## Polynomial and exponential time string functions.
atime_result_string <- atime::atime(
  seconds.limit=0.001,
  N=unique(as.integer(10^seq(0,3,l=100))),
  setup={
    subject <- paste(rep("a", N), collapse="")
    pattern <- paste(rep(c("a?", "a"), each=N), collapse="")
    linear_size_replacement <- paste(rep("REPLACEMENT", N), collapse="")
  },
  PCRE.match=regexpr(pattern, subject, perl=TRUE),
  TRE.match=regexpr(pattern, subject, perl=FALSE),
  constant.replacement=gsub("a","constant size replacement",subject),
  linear.replacement=gsub("a",linear_size_replacement,subject))
plot(atime_result_string)
```

atime_grid

Asymptotic timing grid

Description

Create expressions for asymptotic timing by substituting values into expressions.

Usage

```
atime_grid(
  param.list = list(),
  ...,
  name.value.sep="=",
  expr.param.sep=" ",
  collapse = ",",
  symbol.params=character())
```

Arguments

param.list	Named list of items to replace in ... expressions, default empty list means nothing to replace.
...	Named expressions which each must contain each name of param.list.
name.value.sep	string: separator between names and values from param.list, default "=".
expr.param.sep	string: separator between expressions and parameters, default " ".
collapse	string: separator between parameters, default ",".
symbol.params	character vector: these elements of param.list will be converted to symbols before substitution.

Value

Named list of expressions which can be used as expr.list argument of [atime](#).

Author(s)

Toby Dylan Hocking

Examples

```
data.table::setDTthreads(1) # for CRAN.

## Example 0: with no param.list, same as quote inside named list.
atime::atime_grid(m=mean(data), s=sum(data))
list(m=quote(mean(data)), s=quote(sum(data)))

## Example 1: polynomial vs exponential time regex.
(expr.list <- atime::atime_grid(
  list(PERL=c(TRUE, FALSE)),
```

```

    expr.param.sep="\n",
    regexpr=regexpr(pattern, subject, perl=PERL)))
atime.list <- atime::atime(
  seconds.limit=0.001,
  N=unique(as.integer(10^seq(0,2,l=30))),
  setup={
    subject <- paste(rep("a", N), collapse="")
    pattern <- paste(rep(c("a?", "a"), each=N), collapse="")
  },
  expr.list=expr.list)
plot(atime.list)

## Example 2: symbol.params arg.
sub.param.list <- list(FUN=c("sub","gsub"), PERL=c(TRUE,FALSE))
## with base R we can use as.symbol and substitute:
sapply(sub.param.list$FUN,function(name)substitute(fun("a","",subject), list(fun=as.symbol(name))))
## with atime_grid the analog is to use symbol.params argument:
(sub.expr.list <- atime::atime_grid(
  sub.param.list,
  replace=FUN("a","",subject,perl=PERL),
  symbol.params="FUN"))
sub.atime.list <- atime::atime(
  seconds.limit=0.001,
  setup={
    subject <- paste(rep("a",N),collapse="")
  },
  expr.list=sub.expr.list)
plot(sub.atime.list)

## Customize ggplot, which shows expr.name column as direct labels.
sub.atime.edited <- sub.atime.list
library(data.table)
sub.atime.edited$measurements <- data.table(sub.atime.list$measurements)[
, expr.name := paste0("PERL=",PERL)]
if(require(ggplot2)){
  plot(sub.atime.edited)+
    facet_grid(unit ~ FUN, labeller=label_both, scales="free")
}

```

Description

R package performance testing, by computing time/memory usage of several R expressions of several different data sizes, for several package versions (base, HEAD, CRAN, merge-base, other historical references specified by user). `atime_pkg_test_info` returns an environment containing test code/calls (useful for running a single test), whereas `atime_pkg` runs all tests and saves results/plots to disk.

Usage

```
atime_pkg(pkg.path=".", tests.dir=NULL, verbose=FALSE)
atime_pkg_test_info(pkg.path=".", tests.dir=NULL, verbose=FALSE)
```

Arguments

<code>pkg.path</code>	path to git repository containing R package.
<code>tests.dir</code>	path to directory which contains <code>atime/tests.R</code> , relative to <code>pkg.path</code> (default <code>NULL</code> means first existing directory of <code>"inst"</code> or <code>".ci"</code>).
<code>verbose</code>	logical (default <code>FALSE</code>): print messages?

Details

There should be a `tests.R` code file which defines `test.list`, a list with names corresponding to different tests. Each element should be a list with at least three named elements: `N`, `setup`, `expr`, (possibly others such as `pkg.edit.fun` and `version_name="1234567890abcdef"`) to be passed as named arguments to `atime_versions`, along with the following versions which are passed using the `sha.vec` argument: `base` ref comes from `GITHUB_BASE_REF` environment variable (default `master`), `HEAD` ref is the branch that you want to merge into `base`, `CRAN` is current published version (`sha` value `""`), `merge-base` is most recent common ancestor commit between `base` and `HEAD`. For visualization, default colors are provided for versions with names: `HEAD`, `base`, `CRAN`, `merge-base`, `Before`, `Regression`, `Slow`, `Fixed`, `Fast`; other version names will be gray using the default colors. If `tests.R` defines a variable named `version.colors`, then it should be a character vector of colors to be used instead of the default (names for versions, values for colors). If `tests.R` defines a variable named `pval.thresh`, then it should be a positive numeric value between 0 and 1 (any smaller p-values will generate messages about slowdowns in `HEAD_issues.md`).

Value

`atime_pkg_test_info` returns an environment in which the code of `tests.R` was evaluated, including a variable `test.call` which is a list of un-evaluated `atime_versions` calls, one for each test (use with `eval` to run a single test). `atime_pkg` returns a named list of two elements (`preview` and `all`), each a data table with one row per test shown in result figures. Side effect is that `data/plot` files are saved in `atime` directory, including `tests.RData` (test results which can be read into R if you want to make your own alternative plots/analyses), `tests_all_facet.png` (plot summarizing all test results), `tests_preview_facet.png` (plot summarizing only most significant results), and `install_seconds.txt` (total number of seconds used to install different package versions). Plots emphasize differences between `HEAD` and a comparison version, which is defined as the first of these versions which is available: `merge-base`, `base`, `CRAN`. Plots show test cases, first sorted by `pred.Nx`, which is the ratio of `N` values at the time limit (positive for speedup, negative for slowdown). Ties are broken using `P.value`, un-paired, two-sided, two-sample T-test for difference of mean times at largest `N`.

Author(s)

Toby Dylan Hocking

See Also

[atime_test](#) for defining each test, [atime_test_list](#) for defining common arguments in each element of the test list.

Examples

```
if(FALSE){

  ## Clone binsegRcpp, and checkout a branch which has performance tests.
  tdir <- tempfile()
  dir.create(tdir)
  gert::git_clone("https://github.com/tdhock/binsegRcpp", tdir)
  gert::git_branch_checkout("another-branch", repo=tdir)

  ## Run one test.
  info.env <- atime::atime_pkg_test_info(tdir)
  test_name <- "binseg(1:N,maxSegs=N/2) DIST=l1"
  (one.call <- info.env$test.call[[test_name]])
  one.result <- eval(one.call)
  plot(one.result) # constant difference for small N should be visible.

  ## Run all tests.
  result.list <- atime::atime_pkg(tdir)
  inst.atime <- file.path(tdir, "inst", "atime")
  dir(inst.atime)
  tests.RData <- file.path(inst.atime, "tests.RData")
  (objs <- load(tests.RData))
  plot(pkg.results[[test_name]]) # should look similar.

  atime::atime_versions_remove("binsegRcpp")

}

## https://github.com/tdhock/binsegRcpp/blob/atime-test-funs/.ci/atime/tests.R
## has another real example, see how to run it in tests/testthat/test-versions.R
```

atime_test	<i>Define an atime performance test</i>
------------	---

Description

Use this to define an element of your `test.list` in `atime/tests.R`, prior to running [atime_pkg](#).

Usage

```
atime_test(
  N, setup, expr, times, seconds.limit, verbose,
  pkg.edit.fun, result, setup.version, ...)
```

Arguments

<code>N</code>	numeric vector of data sizes to vary.
<code>setup</code>	expression to evaluate for every data size, before timings, using <code>HEAD</code> .
<code>expr</code>	code with package double-colon prefix, for example <code>PKG::fun(argA, argB)</code> , where <code>PKG</code> is the name of the package specified by <code>pkg.path</code> . This code will be evaluated for each different package version, by replacing <code>PKG:</code> by <code>PKG.SHA:</code> . To run different versions of implicitly-called functions like <code>DT[i, j]</code> , you need to call them explicitly, as in <code>data.table::`[.data.table`(DT, i, j)</code> .
<code>times</code>	number of times to evaluate each timed expression.
<code>seconds.limit</code>	if the median timing of any expression exceeds this many seconds, then no timings for larger <code>N</code> are computed.
<code>verbose</code>	logical, print messages after every data size?
<code>pkg.edit.fun</code>	function called to edit package before installation, should typically replace instances of <code>PKG</code> with <code>PKG.SHA</code> , default works with Rcpp packages.
<code>result</code>	logical, save results? (default <code>FALSE</code>)
<code>setup.version</code>	expression to evaluate for each version, prior to timings.
<code>...</code>	named versions.

Value

List of expressions.

Author(s)

Toby Dylan Hocking

See Also

[atime_test_list](#) for defining common arguments in each element of the test list, [atime_pkg](#) for running tests.

Examples

```
atime::atime_test(
  N=c(1,10),
  setup=data.vec <- rnorm(N),
  expr=binsegRcpp::binseg("mean_norm", data.vec))

## https://github.com/tdhock/binsegRcpp/blob/atime-test-funs/.ci/atime/tests.R
## has a real example, see how to run it in tests/testthat/test-versions.R
```

atime_test_list	<i>Define an atime performance test list</i>
-----------------	--

Description

Use this to define `test.list` in your `atime/tests.R` file, prior to running [atime_pkg](#). Arguments in `...` should all be named; if `name` is an argument of `atime_versions`, it will be copied to each test; otherwise it should be the name of a test.

Usage

```
atime_test_list(
  N, setup, expr, times, seconds.limit,
  verbose, pkg.edit.fun, result, setup.version,
  tests = NULL, ...)
```

Arguments

<code>N</code>	integer vector of data sizes.
<code>setup</code>	expression that depends on <code>N</code> , to run before timings, using <code>HEAD</code> . Not evaluated before copying to each test.
<code>expr</code>	expression to time. Not evaluated before copying to each test.
<code>times</code>	number of times to run <code>expr</code> .
<code>seconds.limit</code>	number of seconds after which we stop trying larger <code>N</code> .
<code>verbose</code>	logical: print output?
<code>pkg.edit.fun</code>	function for editing package prior to testing.
<code>result</code>	logical: save results?
<code>setup.version</code>	expression to evaluate for each version, prior to timings. Not evaluated before copying to each test.
<code>tests</code>	list of tests, with names for tests, values are lists of arguments to pass to atime_versions (combined with <code>...</code>).
<code>...</code>	names for tests, values are lists of arguments to pass to atime_versions (combined with <code>tests</code>).

Value

List representing performance tests, from `...` and `tests`; each element is a list of arguments to pass to [atime_versions](#).

Author(s)

Toby Dylan Hocking

See Also

[atime_test](#) for defining each test, [atime_pkg](#) for running tests.

Examples

```
(test.list.named <- atime::atime_test_list(
  N=as.integer(10^seq(1,3,by=0.5)),
  setup={
    set.seed(1)
    data.vec <- rnorm(N)
  },
  mean_norm=atime::atime_test(expr=binsegRcpp::binseg("mean_norm",data.vec)),
  poisson=atime::atime_test(expr=binsegRcpp::binseg("poisson",data.vec)),
  NULL))

## https://github.com/tdhock/binsegRcpp/blob/atime-test-funs/.ci/atime/tests.R
## has a real example, see how to run it in tests/testthat/test-versions.R
```

atime_versions

Asymptotic timing of git versions

Description

Computation time and memory for a single R expression evaluated using several different git versions.

Usage

```
atime_versions(
  pkg.path, N, setup, expr, sha.vec=NULL,
  times=10, seconds.limit=0.01, verbose=FALSE,
  pkg.edit.fun=pkg.edit.default, result=FALSE,
  N.env.parent=NULL, setup.version=NULL,
  checkout.path=pkg.path,
  ...)
```

Arguments

pkg.path	Path to R package in a git repo.
N	numeric vector of data sizes to vary.
setup	expression to evaluate for every data size, before timings. If one of the versions contains name HEAD, then Package:: is replaced with Package.SHA_for_HEAD:: (in other words, setup is run using the HEAD commit).

expr	code with package double-colon prefix, for example <code>PKG::fun(argA, argB)</code> , where <code>PKG</code> is the name of the package specified by <code>pkg.path</code> . This code will be evaluated for each different package version, by replacing <code>PKG::</code> by <code>PKG.SHA::</code> . To run different versions of implicitly-called functions like <code>DT[i, j]</code> , you need to call them explicitly, as in <code>data.table::`[.data.table`(DT, i, j)</code> .
sha.vec	named character vector / list of versions.
times	number of times to evaluate each timed expression.
seconds.limit	if the median timing of any expression exceeds this many seconds, then no timings for larger <code>N</code> are computed.
verbose	logical, print messages after every data size?
pkg.edit.fun	function called to edit package before installation, should typically replace instances of <code>PKG</code> with <code>PKG.SHA</code> , default works with Rcpp packages that have <code>useDynLib(pkg)</code> in <code>NAMESPACE</code> (no <code>.registration=TRUE</code>).
result	logical or function, passed to atime .
N.env.parent	environment to use as parent of environment created for each data size <code>N</code> , or <code>NULL</code> to use default parent env.
setup.version	expression to evaluate for each version, prior to timings.
checkout.path	path to repo in which we run git checkout prior to package installation. Default is <code>pkg.path</code> , but can be set to a sub-directory with a git submodule.
...	named versions.

Details

For convenience, versions can be specified either as code (...), data (`sha.vec`), or both. Each version should be either "" (to use currently installed version of package, or if missing, install most recent version from CRAN) or a SHA1 hash, which is passed as branch to git checkout; version names used to identify/interpret the output/plots.

Value

list of class `atime` with elements `seconds.limit` (numeric input param), `timings` (data table of results).

Author(s)

Toby Dylan Hocking

See Also

[atime_versions_exprs](#) converts `expr` into a list of expressions, one for each version, passed to [atime](#) as the `expr.list` argument.

Examples

```
if(FALSE){

  tdir <- tempfile()
  dir.create(tdir)
  gert::git_clone("https://github.com/tdhock/binsegRcpp", tdir)
  atime.list <- atime::atime_versions(
    pkg.path=tdir,
    setup={
      max.segs <- as.integer(N/2)
      data.vec <- 1:N
    },
    expr=binsegRcpp::binseg_normal(data.vec, max.segs),
    cv="908b77c411bc7f4fcbcf53759245e738ae724c3e",
    "rm unord map"="dcd0808f52b0b9858352106cc7852e36d7f5b15d",
    mvl_construct="5942af606641428315b0e63c7da331c4cd44c091")
  plot(atime.list)

  atime::atime_versions_remove("binsegRcpp")

  gdir <- tempfile()
  dir.create(gdir)
  gert::git_clone("https://github.com/tdhock/grates", gdir)
  glist <- atime::atime_versions(
    file.path(gdir,"pkg"),
    current = "1aae646888dcedb128c9076d9bd53fcb4075dcda",
    old      = "51056b9c4363797023da4572bde07e345ce57d9c",
    setup    = date_vec <- rep(Sys.Date(), N),
    expr     = grates::as_yearmonth(date_vec))
  plot(glist)

  atime::atime_versions_remove("grates")

}
```

atime_versions_exprs *Create expressions for different git versions*

Description

Install different git commit versions as different packages, then create a list of expressions, one for each version. For most use cases `atime_versions` is simpler, but `atime_versions_exprs` is more flexible for the case of comparing different versions of one expression to another expression.

Usage

```
atime_versions_exprs(
  pkg.path, expr, sha.vec=NULL,
```

```
verbose=FALSE,
pkg.edit.fun=pkg.edit.default,
setup.version=NULL,
checkout.path=pkg.path,
...)
```

Arguments

<code>pkg.path</code>	Path to git repo containing R package.
<code>expr</code>	code with package double-colon prefix, for example <code>PKG::fun(argA, argB)</code> , where <code>PKG</code> is the name of the package specified by <code>pkg.path</code> . This code will be evaluated for each different package version, by replacing <code>PKG::</code> by <code>PKG.SHA::</code> . To run different versions of implicitly-called functions like <code>DT[i, j]</code> , you need to call them explicitly, as in <code>data.table::`[.data.table`(DT, i, j)</code> .
<code>sha.vec</code>	named character vector / list of versions.
<code>verbose</code>	logical, print messages after every data size?
<code>pkg.edit.fun</code>	function called to edit package before installation, should typically replace instances of <code>PKG</code> with <code>PKG.SHA</code> , default works with Rcpp packages, but does not work with all packages. For an example of a custom package editing function, see the <code>atime</code> vignette about <code>data.table</code> .
<code>setup.version</code>	expression to evaluate for each version, prior to timings.
<code>checkout.path</code>	path to repo in which we run git checkout prior to package installation. Default is <code>pkg.path</code> , but can be set to a sub-directory with a git submodule.
<code>...</code>	named versions.

Details

For convenience, versions can be specified either as code (`...`), data (`sha.vec`), or both. Each version should be either `" "` (to use currently installed version of package, or if missing, install most recent version from CRAN) or a SHA1 hash, which is passed as branch arg to git checkout; version names used to identify/interpret the output/plots. Each version is installed as a separate package (to whatever R library is first on `.libPaths()`), using the package name `PKG.SHA`.

Value

A list of expressions, one for each version, created by replacing `PKG::` in `expr` with `PKG.SHA::`. If `setup.version` is used, then each expression has an attribute named `setup.version`, an version-specific expression to run prior to timings. This list can be used as `expr.list` argument of `atime()`, instead of writing code like `atime(name1=Package.SHA1::fun(argA, argB), name2=Package.SHA2::fun(argA, argB))`. If there is a HEAD commit, then the list has attributes `Package` and `new.Package` which are used in [atime_versions](#) to modify setup so that it runs using the HEAD version.

Author(s)

Toby Dylan Hocking

Examples

```
if(FALSE){

  if(requireNamespace("changepoint")){
    tdir <- tempfile()
    dir.create(tdir)
    gert::git_clone("https://github.com/tdhock/binsegRcpp", tdir)
    expr.list <- atime::atime_versions_exprs(
      pkg.path=tdir,
      expr=binsegRcpp::binseg_normal(data.vec, max.segs),
      cv="908b77c411bc7f4fcbcf53759245e738ae724c3e",
      "rm unord map"="dcd0808f52b0b9858352106cc7852e36d7f5b15d",
      "mvl_construct"="5942af606641428315b0e63c7da331c4cd44c091")
    atime.list <- atime::atime(
      N=2^seq(2, 20),
      setup={
        max.segs <- as.integer(N/2)
        data.vec <- 1:N
      },
      expr.list=expr.list,
      changepoint=changepoint::cpt.mean(
        data.vec, penalty="Manual", pen.value=0, method="BinSeg",
        Q=max.segs-1))
    plot(atime.list)
  }

  atime::atime_versions_remove("binsegRcpp")

}
```

atime_versions_remove *Remove packages installed by atime*

Description

atime_versions_exprs installs different git versions of a package, and this function removes them. This can be useful if you have changed compilers, which may have different performance characteristics, so you can then re-compile all package versions using the same compiler.

Usage

```
atime_versions_remove(Package)
```

Arguments

Package	Name of package without SHA.
---------	------------------------------

Details

The library searched is the first on `.libPaths()`.

Value

integer exit status code from unlink, non-zero if removal failed.

Author(s)

Toby Dylan Hocking

glob_find_replace	<i>Find and replace within files</i>
-------------------	--------------------------------------

Description

Find and replace for every file specified by glob.

Usage

```
glob_find_replace(glob, FIND, REPLACE, warn=TRUE)
```

Arguments

glob	character string: glob defining files.
FIND	character string: regex to find.
REPLACE	character string: regex to use for replacement.
warn	logical (default TRUE): warn if nothing replaced.

Value

nothing.

Author(s)

Toby Dylan Hocking

Examples

```
## see vignette("data.table", package="atime")
```

*issue107**atime result for issue 107*

Description

atime was run on windows, to test a speed and memory increase. Memory increases linearly, but the predict method did not return the correct N value for kilobytes=1e4.

Usage

```
data("issue107")
```

Format

List from running atime.

References

<https://github.com/Rdatatable/data.table/issues/7687>

Examples

```
data(issue107, package="atime")
refs <- atime::references_best(issue107)
pred <- predict(refs, kilobytes=1e4)
pred$pred
if(requireNamespace("directlabels"))plot(pred)
```

*issue109**atime results for issue 109*

Description

atime was run on GitHub Actions, to test data.table. One test had a speed improvement, and was shown on the right of the facet plot.

Usage

```
data("issue109")
```

Format

List from running atime_pkg().

References

<https://github.com/Rdatatable/data.table/pull/7401#issuecomment-3457752439>

Examples

```
data(issue109, package="atime")
```

references_best	<i>Best references</i>
-----------------	------------------------

Description

Compute best asymptotic references, for all empirical measurements which are present (not missing) and increasing with data size.

Usage

```
references_best(L, fun.list=NULL)
```

Arguments

L	List output from atime.
fun.list	List of asymptotic complexity reference functions, default NULL means to use package default.

Value

list of class "references_best" with elements references (data table of all references), plot.references (data table of references to show using plot method, default is to show closest larger and smaller references), measurements (data table of measurements).

Author(s)

Toby Dylan Hocking

Examples

```
## Polynomial and exponential time string functions.
atime_result_string <- atime::atime(
  seconds.limit=0.001,
  N=unique(as.integer(10^seq(0,4,l=100))),
  setup={
    subject <- paste(rep("a", N), collapse="")
    pattern <- paste(rep(c("a?", "a"), each=N), collapse="")
    linear_size_replacement <- paste(rep("REPLACEMENT", N), collapse="")
  },
  PCRE.match=regexpr(pattern, subject, perl=TRUE),
```

```

TRE.match=regexpr(pattern, subject, perl=FALSE),
constant.replacement=gsub("a","constant size replacement",subject),
linear.replacement=gsub("a",linear_size_replacement,subject))
(refs_best_string <- atime::references_best(atime_result_string))
## plot method shows each expr and unit in a separate panel.
## default is to show closest larger and smaller references.
plot(refs_best_string)
## modifying plot.references changes violet references shown by plot.
refs_best_string$plot.references <- refs_best_string$ref[c("N", "N^2", "N^3", "2^N"),on="fun.name"]
plot(refs_best_string)
## predict method computes N for given units (default seconds limit).
(pred_string <- predict(refs_best_string))
plot(pred_string)

```

R_CMD_INSTALL

Run R CMD INSTALL

Description

Uses `system()` and `shQuote()` to run R CMD INSTALL, installing the package in `libPaths()[1]`. Can be used in `pkg.edit.fun` passed to [atime_pkg](#), to check if an edited package installs without error.

Usage

```
R_CMD_INSTALL(new.pkg.path)
```

Arguments

`new.pkg.path` path to package

Value

Status code from `system()`, 0 for successful installation, other values for installation errors.

Author(s)

Toby Dylan Hocking

Index

* datasets

issue107, [16](#)

issue109, [16](#)

atime, [2](#), [4](#), [11](#)

atime_grid, [3](#), [4](#)

atime_pkg, [5](#), [7-10](#), [18](#)

atime_pkg_test_info (atime_pkg), [5](#)

atime_test, [7](#), [7](#), [10](#)

atime_test_list, [7](#), [8](#), [9](#)

atime_versions, [9](#), [10](#), [13](#)

atime_versions_exprs, [11](#), [12](#)

atime_versions_remove, [14](#)

glob_find_replace, [15](#)

issue107, [16](#)

issue109, [16](#)

mark, [3](#)

R_CMD_INSTALL, [18](#)

references_best, [17](#)