NAME

sort - perl pragma to control sort() behaviour

SYNOPSIS

```
use sort 'stable'; # guarantee stability
use sort '_quicksort'; # use a quicksort algorithm
use sort '_mergesort'; # use a mergesort algorithm
use sort 'defaults'; # revert to default behavior
no sort 'stable'; # stability not important

use sort '_qsort'; # alias for quicksort

my $current;
BEGIN {
$current = sort::current(); # identify prevailing algorithm
}
```

DESCRIPTION

With the `sort` pragma you can control the behaviour of the builtin `sort()` function.

In Perl versions 5.6 and earlier the quicksort algorithm was used to implement `sort()`, but in Perl 5.8 a mergesort algorithm was also made available, mainly to guarantee worst case O(N log N) behaviour: the worst case of quicksort is O(N**2). In Perl 5.8 and later, quicksort defends against quadratic behaviour by shuffling large arrays before sorting.

A stable sort means that for records that compare equal, the original input ordering is preserved. Mergesort is stable, quicksort is not. Stability will matter only if elements that compare equal can be distinguished in some other way. That means that simple numerical and lexical sorts do not profit from stability, since equal elements are indistinguishable. However, with a comparison such as

```
{ substr($a, 0, 3) cmp substr($b, 0, 3) }
```

stability might matter because elements that compare equal on the first 3 characters may be distinguished based on subsequent characters. In Perl 5.8 and later, quicksort can be stabilized, but doing so will add overhead, so it should only be done if it matters.

The best algorithm depends on many things. On average, mergesort does fewer comparisons than quicksort, so it may be better when complicated comparison routines are used. Mergesort also takes advantage of pre-existing order, so it would be favored for using `sort()` to merge several sorted arrays. On the other hand, quicksort is often faster for small arrays, and on arrays of a few distinct values, repeated many times. You can force the choice of algorithm with this pragma, but this feels heavy-handed, so the subpragmas beginning with a _ may not persist beyond Perl 5.8. The default algorithm is mergesort, which will be stable even if you do not explicitly demand it. But the stability of the default sort is a side-effect that could change in later versions. If stability is important, be sure to say so with a

```
use sort 'stable';
```

The `no sort` pragma doesn't *forbid* what follows, it just leaves the choice open. Thus, after

```
no sort qw(_mergesort stable);
```

a mergesort, which happens to be stable, will be employed anyway. Note that

```
no sort "_quicksort";
no sort "_mergesort";
```

have exactly the same effect, leaving the choice of sort algorithm open.

CAVEATS

As of Perl 5.10, this pragma is lexically scoped and takes effect at compile time. In earlier versions its effect was global and took effect at run-time; the documentation suggested using `eval()` to change the behaviour:

```
{ eval 'use sort qw(defaults _quicksort)'; # force quicksort
eval 'no sort "stable"'; # stability not wanted
print sort::current . "\n";
@a = sort @b;
eval 'use sort "defaults"'; # clean up, for others
}
{ eval 'use sort qw(defaults stable)'; # force stability
print sort::current . "\n";
@c = sort @d;
eval 'use sort "defaults"'; # clean up, for others
}
```

Such code no longer has the desired effect, for two reasons. Firstly, the use of `eval()` means that the sorting algorithm is not changed until runtime, by which time it's too late to have any effect. Secondly, `sort::current` is also called at run-time, when in fact the compile-time value of `sort::current` is the one that matters.

So now this code would be written:

```
{ use sort qw(defaults _quicksort); # force quicksort
no sort "stable"; # stability not wanted
my $current;
BEGIN { $current = sort::current; }
print "$current\n";
@a = sort @b;
# Pragmas go out of scope at the end of the block
}
{ use sort qw(defaults stable); # force stability
my $current;
BEGIN { $current = sort::current; }
print "$current\n";
@c = sort @d;
}
```