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 = 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

This pragma is not lexically scoped: its effect is global to the program it appears in. That means the following will probably not do what you expect, because *both* pragmas take effect at compile time, before *either* `sort()` happens.

```
{ use sort "_quicksort";
  print sort::current . "\n";
  @a = sort @b;
}
{ use sort "stable";
  print sort::current . "\n";
  @c = sort @d;
}
# prints:
# quicksort stable
# quicksort stable
```

You can achieve the effect you probably wanted by using `eval()` to defer the pragmas until run time. Use the quoted argument form of `eval()`, *not* the BLOCK form, as in

```
eval { use sort "_quicksort" }; # WRONG
```

or the effect will still be at compile time. Reset to default options before selecting other subpragmas (in case somebody carelessly left them on) and after sorting, as a courtesy to others.

```
{ 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
}
# prints:
# quicksort
# stable
```

Scoping for this pragma may change in future versions.