---

set sortmethod — Specify a sort method

Description

The default sort method was changed in Stata 17 to offer faster performance. The new method is designated fsort. Prior to Stata 17, the method was qsort. set sortmethod lets you explicitly choose between the two methods.

Syntax

set sortmethod { default | fsort | qsort }

fsort is the current, faster sort.

qsort is the sort prior to Stata 17.

default is whichever of fsort or qsort was the default for the currently set user version.

Remarks and examples

Remarks are presented under the following headings:

> *Overview and version control*
> *Controlling the sorter within a program*
> *Reproducibility*

Overview and version control

The default sort method was changed in Stata 17 to offer improved sort performance. The new sorter is a modified quicksort with a three-way partition and switches to an insertion sort algorithm when the problem size is less than 8. This method is designated fsort. Prior to Stata 17, the method was qsort—a standard quicksort algorithm.

sort follows user version control, so all existing programs and ado-file commands will automatically use the new faster fsort regardless of the version specified in the program. Conversely, version-controlled do-files will use whichever sort was the default at the time of the version specified in the do-file. Likewise, interactive use of sort will use whichever version was the default for the current version.

Controlling the sorter within a program

In the unlikely case that you want to require the straight quicksort be used in a program, add set sortmethod qsort to the program. Be aware that setting the sortmethod is a global action. All future sorts will use the old qsort program, both within the program and after the program terminates.

The safest way to force the method to be qsort within a program is to save the current setting in your program and reset that setting when your sorts are complete. Do all this while being sure that your program does not exit before you get the setting restored. Here is the safest construct that protects against both errors and user-entered break keys.

```
nobreak {
        local holdsortmeth = c(sortmethod)
        capture noisily {
                ... your code ...
        }
        local rc = _rc
        set sortmethod 'holdsortmeth'
}

if ('rc')  exit 'rc'
```

Reproducibility

If your sort keys produce a unique ordering of the data, your results are obviously and automatically reproducible. Every time you sort on those keys in any version of Stata or in any flavor of Stata, you will get the same results. And that is truly the right way to address reproducible sorts.

If you are sorting on keys that do not produce a unique sort but instead have observations with tied values of all the sort variables, you should think long and hard about why you want such an indeterminate ordering. We can think of no case where you would want to perform a sort that does not produce a unique ordering. See *Sorting with ties* in [D] **sort** for a discussion of creating unique sort keys for your sort, even when you want the ties broken randomly.

That said, Stata allows you to perform sorts with ties that cannot lead to a unique ordering. Moreover, that ordering will not be the same when you rerun the same sort command. If you have ties in the variable xyz and type

    . sort xyz

And then, after some other commands have changed the sort order, again type

    . sort xyz

The orderings of the data after each of those commands will be different!

Both the fsort and qsort methods require an initial jumbling of the order of the data to avoid potentially severe speed penalties for some specific initial orderings. This jumbling is done pseudorandomly using a fast random-number generator. This is not the excellent random-number generator used in all of Stata's random-number functions; see [R] **set seed** for the specifics. The jumbler's purpose is to be fast, not to have good properties in the pseudo–random numbers that are generated. What's more, the ordering will differ across Stata/SE and Stata/MP, when Stata/MP is set to use multiple cores, almost always. Again, this comes down to performance. Stata/MP performs the initial jumbling in parallel. What's more, methods fsort and qsort also use the initial jumbling in different ways and will produce different orderings of tied observations.

So if you want reproducible orderings, do not sort on variables with tied values in some observations. If you want to break the ties randomly, create a random number using runiform(), and sort on it. Again, see *Sorting with ties* in [D] **sort**.

Also see

[P] **creturn** — Return c-class values

[P] **set sortrngstate** — Set the state of sort's randomizer

[D] **sort** — Sort data

[R] **set** — Overview of system parameters

For suggested citations, see the FAQ on citing Stata documentation.