Sorting Algorithms — Stability

Let A be an array, and let $<$ be a strict weak ordering on the elements of A.

A sorting algorithm is ***stable*** if

$$i < j \text{ and } A[i] \sim A[j] \text{ implies } \pi(i) < \pi(j).$$

where π is the sorting permutation (sorting moves $A[i]$ to position $\pi(i)$.)

Informally, stability means that equivalent elements retain their relative positions, after sorting. (If the elements have class/structure type and are ordered according to their value on one field — the *key* field — then *equivalent* elements are elements with equal keys.)

Suppose we apply a sorting algorithm with this array as input, using the state (`NY`, `IL`, etc) as the key.

New York	NY
Chicago	IL
Detroit	MI
Buffalo	NY
Milwaukee	WI
Champaign	IL

There are four correct outputs.

Chicago	IL
Champaign	IL
Detroit	MI
New York	NY
Buffalo	NY
Milwaukee	WI

Champaign	IL
Chicago	IL
Detroit	MI
New York	NY
Buffalo	NY
Milwaukee	WI

Chicago	IL
Champaign	IL
Detroit	MI
Buffalo	NY
New York	NY
Milwaukee	WI

Champaign	IL
Chicago	IL
Detroit	MI
Buffalo	NY
New York	NY
Milwaukee	WI

But for a *stable* sorting algorithm, *only the first output is correct*.

Chicago IL comes before **Champaign IL** in the *original* array implies **Chicago IL** comes before **Champaign IL** in the *sorted* array

New York NY comes before **Buffalo NY** in the *original* array implies **New York NY** comes before **Buffalo NY** in the *sorted* array

In practice, we sometimes need a stable sorting algorithm.

We might need sort the array A above alphabetically by state, with all cities in a single state appearing in alphabetical order.

Our sorting software might allow sorting on only one field at a time.

We may 1) First sort the array A alphabetically by city.

2) Then sort the array A alphabetically by state, using a stable sorting algorithm.

Chicago	IL
Champaign	IL
Detroit	MI
New York	NY
Buffalo	NY
Milwaukee	WI
Albany	NY
Green Bay	WI
Syracuse	NY
Rockford	IL
Evanston	IL

Sort alphabetically by city.

Albany	NY
Buffalo	NY
Champaign	IL
Chicago	IL
Detroit	MI
Evanston	IL
Green Bay	WI
Milwaukee	WI
New York	NY
Rockford	IL
Syracuse	NY

Stable sort alphabetically by state.

Champaign	IL
Chicago	IL
Evanston	IL
Rockford	IL
Detroit	MI
Albany	NY
Buffalo	NY
New York	NY
Syracuse	NY
Green Bay	WI
Milwaukee	WI

Unfortunately, many of the (otherwise) best sorting algorithms are not stable.

For example, *quicksort* and *heapsort* are not stable. (*Mergesort* property implemented is stable.)

Any sorting algorithm may be made stable, at a price: The price is Θ(n) extra space, and moderately increased running time (less than doubled, most likely).

We have to (temporarily) append a sequence number to the key of each element of the array. The sequence number serves as a tie-breaker.

Chicago	IL
Champaign	IL
Detroit	MI
New York	NY
Buffalo	NY
Milwaukee	WI
Albany	NY
Green Bay	WI
Syracuse	NY
Rockford	IL
Evanston	IL

Temporarily append sequence number to key fields →

Chicago	IL01
Champaign	IL02
Detroit	MI03
New York	NY04
Buffalo	NY05
Milwaukee	WI06
Albany	NY07
Green Bay	WI08
Syracuse	NY09
Rockford	IL10
Evanston	IL11

Sorting this array, with the (modified) state field as the key, using any sorting algorithm, produces a stable sort of the original array.

The C++ standard library provides a choice between a stable sorting template function **`stable_sort()`**, and a (presumably faster) non-stable sorting template function **`sort()`**.

The Java library (class **`java.util.Arrays`**) provides static methods for sorting objects, that are guaranteed to be stable.