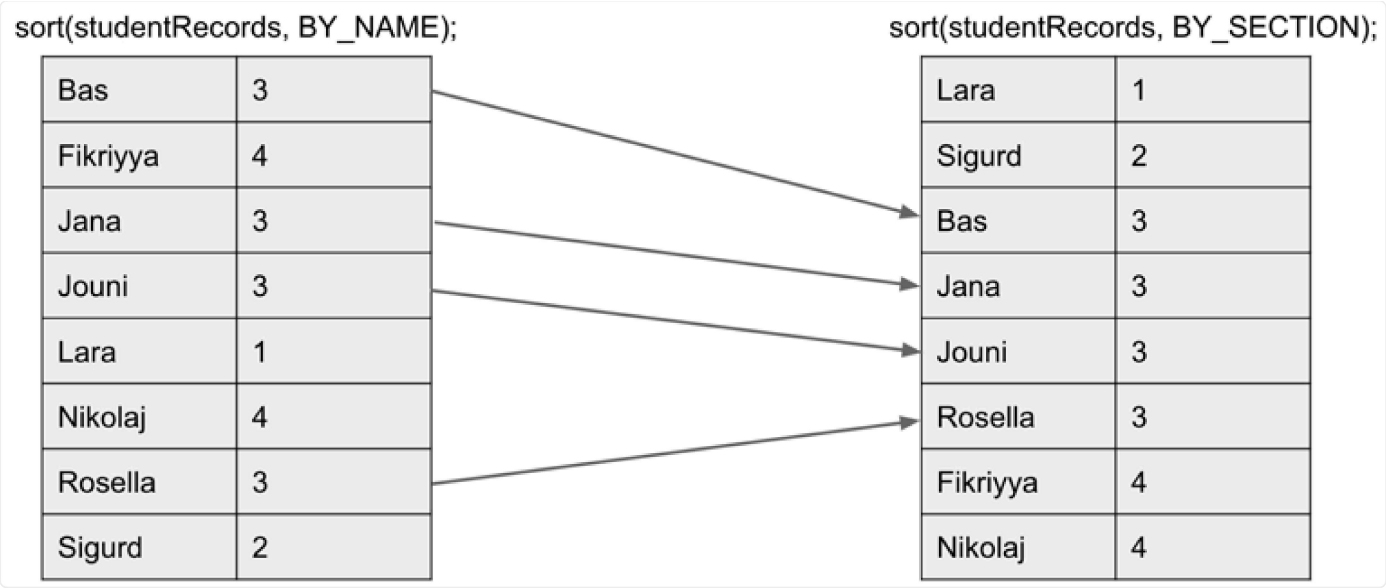
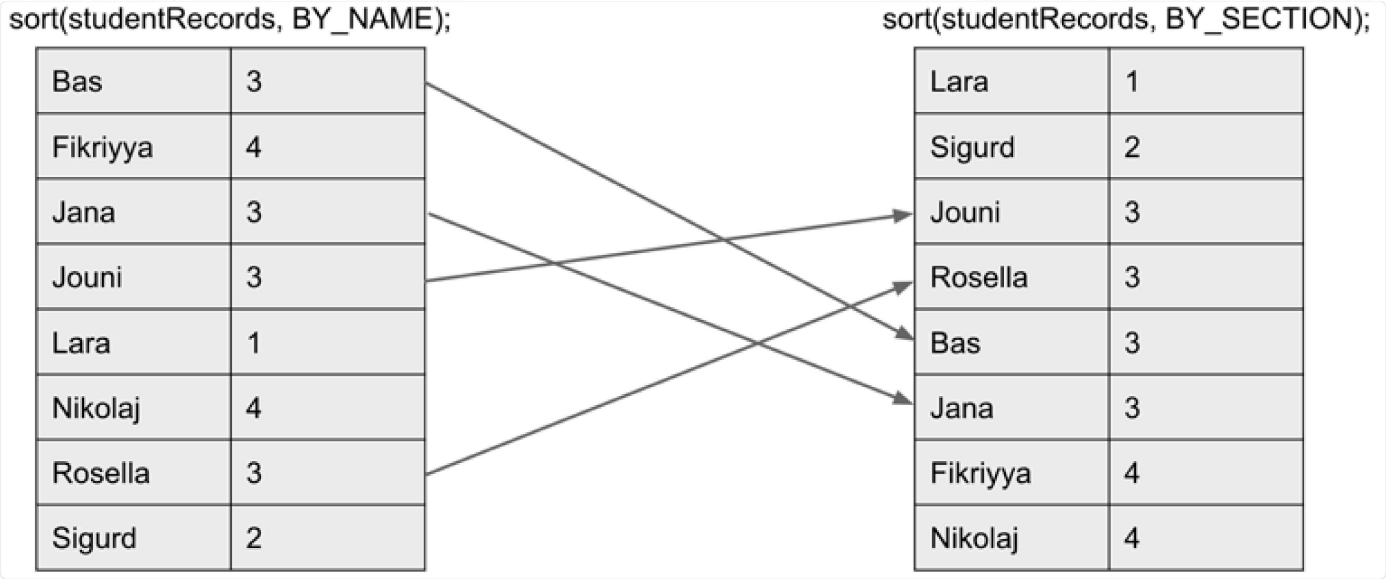
32.3 Stability, Adaptiveness, and Optimization

Stability

A sort is stable if the order of equivalent elements is preserved. The following is an example of a stable sort. After sorting by section, notice how Bas, Jana, Jouni, and Rosella are in the same order as before sorting. If we want records sorted by section and then by name within each section, we can sort by name and then by section as below.



The following example is an unstable sort. It can make things really annoying! If we want records sorted by section and then by name within each section, we can't just sort by name and then by section as before. After an unstable sort, the previous ordering is not maintained.



Are some of the sorts we learned stable? Insertion sort is stable! Equivalent elements move past their equivalent brethren. MergeSort is stable. HeapSort is not stable. QuickSort can be stable depending on its partitioning scheme, but its stability cannot be assumed since many of its popular partitioning schemes, like Hoare, are unstable.

Optimizations

Adaptiveness - A sort that is adaptive exploits the existing order of the array. Examples are InsertionSort, SmoothSort, and TimSort.

Switch to Insertion Sort - When a subproblem reaches size 15 or lower, use insertion sort. It is very very fast for inputs of small sizes.

Exploit restrictions on set of keys - For example, if the number of keys is some constant, we can use this constraint to sort faster by applying 3-way QuickSort.

Switch from QuickSort - If the recursion goes too deep, switch to a different type of sort.

Previous
32.2 Quick Select

Next
32.4 Summary

Last updated 1 year ago