in-place sorting algorithm

Canonical name	InplaceSortingAlgorithm
Date of creation	2013-03-22 11:44:31
Last modified on	2013-03-22 11:44:31
Owner	archibal (4430)
Last modified by	archibal (4430)
Numerical id	10
Author	archibal (4430)
Entry type	Definition
Classification	msc 68P10
Classification	msc 16L30
Synonym	in-situ sorting algorithm
Related topic	SortingProblem
Related topic	TotalOrder
Related topic	Bubblesort

A sorting algorithm is said to be *in-place* if it requires very little additional space besides the initial array holding the elements that are to be sorted. Normally "very little" is taken to mean that for sorting n elements, $O(\log n)$ extra space is required.[1] This is reasonable because in a purely mathematical analysis of the algorithms, *any* sorting algorithm that operates on a contiguous array requires $O(\log n)$ extra space, since this is the number of bite required to represent an index into the array. Normally one ignores this as in most algorithms one treats integers as a data type requiring constant space and constant time for basic operations. An exception are number theoretic algorithms often encountered in http://planetmath.org/NumberTheoryAndCryptographycryptograph

Of course, the efficiency of any algorithm will depend on how the data is stored; usually, the elementary discussion of sorting algorithms focuses on sorting elements stored in a contiguous array (which has constant-time access and swap operations but takes a long time to do shifts). In this context, Heapsort is an in-place sorting algorithm, since it requires constant additional space. Quicksort is also considered in-place, although it requires an amount of extra space logarithmic in the size of the array. Most implementations of Quicksort are recursive, and each recursive call needs to store some local variables on the stack; the depth of the recursion is usually $O(\log n)$, but can be $O(n)$ in degenerate cases. If you try to convert Quicksort to a non-recursive algorithm, you will find that it is necessary to store some intermediatde indexes on a stack, which usually grows up to size $O(\log n)$ but may grow up to size $O(n)$. Mergesort is a sorting algorithm with running time $O(n \log n)$ which is not in-place when dealing with arrays.

On the other hand, if one is sorting linked lists, looking up a general element by index requires $O(n)$ steps, so Quicksort and Heapsort need to be drastically modified to run in even $O(n^2)$; this is not a natural context to use these algorithms. Mergesort, on the other hand, retains its $O(n \log n)$ time complexity and requires only $O(\log n)$ extra space.

In-place sorting is often useful when dealing with truly enormous data sets, where $O(n)$ extra space is truly difficult to work with. In-place sorting algorithms may or may not have better locality of reference than other sorting algorithms. Truly enormous data sets are usually stored on media where random access of data is very expensive but extra storage space is

[1]In fact, one should really allow any polynomial in $\log n$ in order for to qualify. So perhaps the condition should be expressed as $O(n^\epsilon)$ for every $\epsilon > 0$. These are distinctions in complexity that are almost always ignored since they are dwarfed in practical problems by implementation differences.

realtively inexpensive (such as disks); in these cases, whether an algorithm is in-place is less relevant. In fact, even when dealing with data stored as arrays Mergesort is much more efficient for disk-based sorting than the other algorithms because of its better locality of reference.

It should be pointed out in any analysis of the standard sorting algorithms that they are based on an assumption that is almost never true: they assume that the only operation possible on keys is comparison. Sorting methods taking advantage of the structure of keys (say, they are strings) can be much faster both asymptotically and in practice; they are generally not in-place, but the extra space is often worth the time advantage.