Introsort

Introsort or **introspective sort** is a hybrid sorting algorithm that provides both fast average performance and (asymptotically) optimal worst-case performance. It begins with quicksort and switches to heapsort when the recursion depth exceeds a level based on (the logarithm of) the number of elements being sorted. This combines the good parts of both algorithms, with practical performance comparable to quicksort on typical data sets and worst-case O(n log n) runtime due to the heap sort. Since both algorithms it uses are comparison sorts, it too is a comparison sort.

Introsort

Class	Sorting algorithm
Data structure	Array
Worst-case performance	O(n log n)
Average performance	O(n log n)

Introsort was invented by David Musser in Musser (1997), in which he also introduced introselect, a hybrid selection algorithm based on quickselect (a variant of quicksort), which falls back to median of medians and thus provides worst-case linear complexity, which is optimal. Both algorithms were introduced with the purpose of providing generic algorithms for the C++ Standard Library which had both fast average performance and optimal worst-case performance, thus allowing the performance requirements to be tightened.[1]

Contents

Pseudocode

If a heapsort implementation and partitioning functions of the type discussed in the quicksort article are available, the introsort can be described succinctly as

```
procedure sort(A : array):
    let maxdepth = ⌊log(length(A))⌋ × 2
    introsort(A, maxdepth)

procedure introsort(A, maxdepth):
    n ← length(A)
    p ← partition(A)  // assume this function does pivot selection, p is the final position of the pivot
    if n ≤ 1:
        return  // base case
    else if p > maxdepth:
        heapsort(A)
    else:
        introsort(A[0:p], maxdepth - 1)
        introsort(A[p+1:n], maxdepth - 1)
```

The factor two in the maximum depth is arbitrary; it can be tuned for practical performance. $A[i{:}j]$ denotes the array slice of items i to j.

Analysis

In quicksort, one of the critical operations is choosing the pivot: the element around which the list is partitioned. The simplest pivot selection algorithm is to take the first or the last element of the list as the pivot, causing poor behavior for the case of sorted or nearly sorted input. Niklaus Wirth's variant uses the middle element to prevent these occurrences, degenerating to $O(n^2)$ for contrived sequences. The median-of-3 pivot selection algorithm takes the median of the first, middle, and last elements of the list; however, even though this performs well on many real-world inputs, it is still possible to contrive a *median-of-3 killer* list that will cause dramatic slowdown of a quicksort based on this pivot selection technique.

Musser reported that on a median-of-3 killer sequence of 100,000 elements, introsort's running time was 1/200 that of median-of-3 quicksort. Musser also considered the effect on caches of Sedgewick's delayed small sorting, where small ranges are sorted at the end in a single pass of insertion sort. He reported that it could double the number of cache misses, but that its performance with double-ended queues was significantly better and should be retained for template libraries, in part because the gain in other cases from doing the sorts immediately was not great.

Implementations

Introsort or some variant is used in a number of standard library sort functions, including some C++ sort implementations.

The June 2000 SGI C++ Standard Template Library stl_algo.h (http://www.sgi.com/tech/stl/stl_algo.h) implementation of unstable sort uses the Musser introsort approach with the recursion depth to switch to heapsort passed as a parameter, median-of-3 pivot selection and the Knuth final insertion sort pass for partitions smaller than 16.

The GNU Standard C++ library is similar: uses introsort with a maximum depth of $2 \times \log_2 n$, followed by an insertion sort on partitions smaller than 16.[2]

The Microsoft .NET Framework Class Library, starting from version 4.5 (2012), uses Introsort instead of simple QuickSort.[3]

References

1. "Generic Algorithms (http://www.cs.rpi.edu/~musser/gp/algorithms.html)", David Musser
2. libstdc++ Documentation: Sorting Algorithms (https://gcc.gnu.org/onlinedocs/libstdc++/libstdc++-html-USERS-4.4/a01027.html)
3. Array.Sort Method (Array) (http://msdn.microsoft.com/en-us/library/6tf1f0bc(v=vs.110).aspx)

General

- Musser, David R. (1997). "Introspective Sorting and Selection Algorithms" (http://www.cs.rpi.edu/~musser/gp/introsort.ps). *Software: Practice and Experience*. Wiley. **27** (8): 983–993. doi:10.1002/(SICI)1097-024X(199708)27:8<983::AID-SPE117>3.0.CO;2-# (https://doi.org/10.1002%2F%28SICI%291097-024X%28199708%2927%3A8%3C983%3A%3AAID-SPE117%3E3.0.CO%3B2-%23) (inactive 2017-08-27).
- Niklaus Wirth. "Algorithms and Data Structures". Prentice-Hall, Inc., 1985. ISBN 0-13-022005-1.