Stable, efficient sort?

Asked 16 years, 3 months ago Modified 12 years, 9 months ago

Viewed 6k times

14

I'm trying to create an unusual associative array implementation that is very space-efficient, and I need a sorting algorithm that meets all of the following:

1. Stable (Does not change the relative ordering of elements with equal keys.)
2. In-place or almost in-place (O(log n) stack is fine, but no O(n) space usage or heap allocations.
3. O(n log n) time complexity.

Also note that the data structure to be sorted is an array.

It's easy to see that there's a basic algorithm that matches any 2 of these three (insertion sort matches 1 and 2, merge sort matches 1 and 3, heap sort matches 2 and 3), but I cannot for the life of me find anything that matches all three of these criteria.

algorithm language-agnostic data-structures sorting

Share

Improve this question

Follow

edited Sep 22, 2008 at 3:26

Nescio

28.4k ● 10 ● 55 ● 76

asked Sep 22, 2008 at 3:23

dsimcha

68.6k ● 55 ● 219 ● 340

1 Will your data have regular updates? If so then putting in one huge array is a bad idea. Consider a structure that can be fragmented such as a B-tree or rope. – finnw Oct 4, 2008 at 14:35

It seems odd to be happy with O(n log n) time complexity but have an issue with O(n) space usage.. Could you elaborate on what your actual objective is? there's a risk you are falling into the XY problem trap. – mikera Jan 27, 2012 at 15:51

13 Answers

Sorted by: Highest score (default)

Merge sort can be written to be in-place I believe. That may be the best route.

10

Share Improve this answer

Follow

edited Sep 23, 2008 at 20:33

answered Sep 22, 2008 at 3:25

jjnguy

139k ● 53 ● 297 ● 326

comjnl.oxfordjournals.org/cgi/content/abstract/35/6/643 This is probably the algorithm you want. – Corey D Aug 17, 2009 at 20:29

9

Note: standard quicksort is *not* O(n log n) ! In the worst case, it can take up to O(n^2) time. The problem is that you might pivot on an element which is far from the median, so that your recursive calls are highly unbalanced.

There is a way to combat this, which is to carefully pick a median which is guaranteed, or at least very likely, to be close to the median. It is surprising that you can actually find the exact median in linear time, although in your case it sounds like you care about speed so I would not suggest this.

I think the most practical approach is to implement a **stable quicksort** (it's easy to keep stable) but use the **median of 5 random values as the pivot** at each step. This makes it highly unlikely that you'll have a slow sort, and is stable.

By the way, merge sort can be done in-place, although it's tricky to do both in-place and stable.

Share Improve this answer

Follow

answered Sep 22, 2008 at 4:14

Tyler

28.9k ● 12 ● 93 ● 108

1 Fundamentals of Algorithms pg 237 describes a way to make quicksort O(n log n) *except* if all elements are the same. It recursively picks the median to pivot on, returning the pivoted list which quicksort then recurses down. Having said that, I agree that the median of 5 is the best way to do it.
– Michael Deardeuff Sep 24, 2008 at 18:22

3

What about quicksort?

Exchange can do that too, might be more "stable" by your terms, but quicksort is faster.

Share Improve this answer

Follow

answered Sep 22, 2008 at 3:25

davenpcj
12.7k ● 5 ● 42 ● 38

1 The example given in en.wikipedia.org/wiki/Quicksort#Algorithm is stable, though not the most efficient version of qsort. – freespace Sep 22, 2008 at 3:29

It's my understanding that variations of Quicksort can be made stable, or efficient, but not both at the same time. – cjm Sep 22, 2008 at 3:49

3

There's a list of sort algorithms on Wikipedia. It includes categorization by execution time, stability, and allocation.

Your best bet is probably going to be modifying an efficient unstable sort to be stable, thereby making it less efficient.

Share Improve this answer

Follow

answered Sep 22, 2008 at 4:07

davenpcj

12.7k ● 5 ● 42 ● 38

2

There is a class of stable in-place merge algorithms, although they are complicated and linear with a rather high constant hidden in the O(n). To learn more, have a look at this article, and its bibliography.

Edit: the merge phase is linear, thus the mergesort is nlog_n.

Share Improve this answer

Follow

answered Sep 22, 2008 at 8:32

Rafał Dowgird

45k ● 11 ● 79 ● 94

2

Because your elements are in an array (rather than, say, a linked list) you have some information about their original order available to you in the array indices themselves. You can take advantage of this by writing your sort and comparison functions to be aware of the indices:

```
function cmp( ar, idx1, idx2 )
{
   // first compare elements as usual
   rc = (ar[idx1]<ar[idx2]) ? -1 : (
(ar[idx1]>ar[idx2]) ? 1 : 0 );

   // if the elements are identical, then compare
their positions
   if( rc != 0 )
      rc = (idx1<idx2) ? -1 : ((idx1>idx2) ? 1 :
0);

   return rc;
}
```

This technique can be used to make any sort stable, as long as the sort ONLY performs element swaps. The indices of elements will change, but the relative order of identical elements will stay the same, so the sort remains robust. It won't work out of the box for a sort like heapsort because the original heapification "throws away" the relative ordering, though you might be able to adapt the idea to other sorts.

Share Improve this answer

Follow

answered Oct 4, 2008 at 14:26

Eric

11.6k ● 14 ● 60 ● 100

I was going to propose the same thing. – Konrad Rudolph Oct 4, 2008 at 14:45

1 This won't work for all algorithms. A sort could compare a_1 with some b, causing it to get swapped relative to some a_2 between them. You may be able to use it for some, but you have a hefty proof obligation. – wnoise Oct 10, 2008 at 4:51

2

Quicksort can be made stable reasonably easy simply by having an sequence field added to each record, initializing it to the index before sorting and using it as the least significant part of the sort key.

This has a slightly adverse effect on the time taken but it doesn't affect the time complexity of the algorithm. It also has a minimal storage cost overhead for each record, but that rarely matters until you get very large numbers of records (and is mimimized with larger record sizes).

I've used this method with `C`'s `qsort()` function to avoid writing my own. Each record has a 32-bit integer added and populated with the starting sequence number before calling `qsort()`.

Then the comparison function checked the keys *and* the sequence (this guarantees there are no duplicate keys), turning the quicksort into a stable one. I recall that it still outperformed the inherently stable mergesort for the data sets I was using.

Your mileage may vary, so always remember: *Measure, don't guess!*

Share Improve this answer Follow

edited Aug 14, 2009 at 12:42

answered Feb 25, 2009 at 11:16

paxdiablo
880k • 241 • 1.6k • 2k

2

Quicksort can be made stable by doing it on a linked list. This costs n to pick random or median of 3 pivots but with a very small constant (list traversal).

By splitting the list and ensuring that the left list is sorted so same values go left and the right list is sorted so the same values go right, the sort will be implicity stable for no real extra cost. Also, since this deals with assignment rather than swapping, I think the speed might actually be slightly better than a quick sort on an array since there's only a single write.

So in conclusion, list up all your items and run quicksort on a list

Share Improve this answer

Follow

answered Nov 4, 2009 at 16:51

nik3daz
111 • 1 • 7

1

Don't worry too much about O(n log n) until you can demonstrate that it matters. If you can find an O(n^2) algorithm with a drastically lower constant, go for it!

The general worst-case scenario is not relevant if your data is highly constrained.

In short: Run some test.

Share Improve this answer

Follow

answered Sep 23, 2008 at 15:15

I answer wrong - have fun

2,811 ● 2 ● 26 ● 36

2 I agree with phyzome in general, big-O doesn't matter unless N has a decent chance of being large. However, what I'm trying to do is write a space-efficient associative array to fit large amounts of data in RAM, so the whole point is that N is huge. – dsimcha Sep 23, 2008 at 23:39

1

There's a nice list of sorting functions on wikipedia that can help you find whatever type of sorting function you're after.

For example, to address your specific question, it looks like an in-place merge sort is what you want.

However, you might also want to take a look at strand sort, it's got some very interesting properties.

Share Improve this answer

Follow

answered Aug 17, 2009 at 20:04

ReaperUnreal

980 ● 7 ● 20

1

I have implemented a stable in-place quicksort and a stable in-place merge sort. The merge sort is a bit faster, and guaranteed to work in O(n*log(n)^2), but not the quicksort. Both use O(log(n)) space.

Share Improve this answer Follow

edited Mar 2, 2012 at 11:02

answered Dec 9, 2011 at 15:11

Thomas Mueller
50k ● 15 ● 120 ● 134

By the way, it might be possible to create more than two partitions. Also, smaller arrays should be sorted with a different algorithm (for example insertion sort). The algorithm above is just a starting point really. – Thomas Mueller Dec 9, 2011 at 16:45

0

Perhaps shell sort? If I recall my data structures course correctly, it tended to be stable, but it's worse case time is O(n log^2 n), although it performs O(n) on almost sorted data. It's based on insertion sort, so it sorts in place.

Share Improve this answer Follow

edited Sep 22, 2008 at 18:20

answered Sep 22, 2008 at 3:27

Ryan
15.3k ● 7 ● 50 ● 50

5 So it's sometimes stable? I think that is the exact definition of unstable :) – leppie Feb 25, 2009 at 11:17

Sometimes is different than usually :) – Ryan Feb 25, 2009 at 18:06

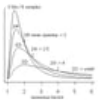
0

Maybe I'm in a bit of a rut, but I like hand-coded merge sort. It's simple, stable, and well-behaved. The additional temporary storage it needs is only `N*sizeof(int)`, which isn't too bad.

Share Improve this answer

Follow

answered Aug 17, 2009 at 19:44



Mike Dunlavey

40.6k ● 15 ● 94 ● 138