

Re: attempt at qsort implementation

Source: <http://coding.derkeiler.com/Archive/C/ CPP/comp.lang.c/2004-09/2968.html>

From: Gordon Burditt (gordonb.jgi3m_at_burditt.org)

Date: 09/29/04

Date: 29 Sep 2004 06:01:55 GMT

*>I had some spare time and decided to try to implement the standard library
>qsort. This is a pretty simpleminded attempt, but it seems to be working. I
>have to point out that efficiency is not at all an issue for me, and this is
>purely a toy, so to speak. I'm quite aware that this is not the quickest way
>to implement the quicksort algorithm, but I basically just wanted to try to
>make a "generic function".*

qsort() is not required to implement any particular sort algorithm. It's just supposed to sort. It's not specified HOW. A bubble sort is not an invalid implementation. A recursive bogosort [*] (which operates in time equivalent to $O(e^{**infinity})$) is probably an invalid implementation since it NEVER finishes.

*>I tested it with a few arrays of structures
>containing all sorts of types (this, ofcourse, doesn't mean that it's
>correct, even by a long shot :). I would appreciate any and all constructive
>comments. The code below is a snippet. All the necessary headers are
>included and uchar is a typedef for unsigned char.
>Thank you.*

If `count == 1`, the algorithm does a compare anyway. I know you weren't going for real efficiency here, but this seems a bit strange.

If `sz == 0`, various wierd and obnoxious things will happen, but the caller IS asking for it.

It seems like this algorithm could end up recursing a lot. This could be a problem with large amounts of data.

One thing worth checking here is how well your algorithm will operate with a bogus compare routine (e.g. one which always returns `-1`, or one for which `cmp(a,b)` is not guaranteed to equal `-cmp(b,a)`). Some implementations end up going outside the passed-in array in such circumstances.

[*] Technical note: a bogosort operates by creating all possible

comp.lang.c: Re: attempt at qsort implementation

orders of the sorted array, determining how many elements are out of order, and returns the order with the least number of elements out of order by sorting them and returning the first one. In other words, it turns the problem of sorting N elements into one of sorting N! elements.

A recursive bogosort uses bogosort to sort the possible orders by number of elements out of order, thereby turning the problem of sorting N elements into one of sorting N!!!!!!!!!!!!!!!!!!!!!!!!!!!!..... elements. Bogosorts typically include other performance-wasting code such as:

```
while (malloc(INT_MAX)) fork();
```

since they will never finish anyway.

Gordon L. Burditt