

## Re: Sorting records using sort()

**Source:** [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.cpp/2004-01/0286.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.cpp/2004-01/0286.html)

---

**From:** Elijah Bailey ([geomrock\\_at\\_hotmail.com](mailto:geomrock_at_hotmail.com))

**Date:** 01/04/04

Date: Sun, 4 Jan 2004 04:15:33 +0000 (UTC)

m@remove.this.part.rtij.nl (Martijn Lievaart) wrote in message  
news:<pan.2004.01.02.12.56.21.625773@remove.this.part.rtij.nl>...

> On Fri, 02 Jan 2004 03:13:26 -0800, Elijah Bailey wrote:

>

> [ Please don't top post, thank you. M4 ]

>

> [ Crossposted to csc++, is this standard compliant? ]

>

> [ Short description of the problem ]

>

> We have an array of  $n*m$  bytes. It holds  $n$  objects of size  $m$ . Both are only  
> known at runtime. For some strange reason we want to sort this using  
> `std::sort`. We know there are other solutions, the question is, can it be  
> done using `std::sort`.

>

Thanks Martijn. This is a much better description than i came up with.

> Constraints: Not clear, but memory seems to be an issue, creating an array  
> of pointers and sort that is out.

Constraints:  $O(n)$  space is not allowed. So you can not create a  
pointer

to each object and sort them.  $O(\log(n))$  space is perfectly ok.

$O(\log(n)*m)$  is also allowed if that simplifies things.

My data has  $n \gg m$ , i.e.  $n$  is much greater than  $m$ .

>

>> "Ron Natalie" <[ron@sensor.com](mailto:ron@sensor.com)> wrote in message

>> news:<3ff4297b\$0\$31857\$9a6e19ea@news.newshosting.com>...

>>> "Jeff Schwab" <[jeffplus@comcast.net](mailto:jeffplus@comcast.net)> wrote in message

>>> news:WoidnaAaasOi8mmiRVn-hA@comcast.com...

>>>

>>>> Writing your own iterators would not be trivial, but probably would

>>>> be a educational. Having such iterators available would make your

>>>> data structure compatible with most of the standard library, and

>>>> might avoid countless future headaches.

comp.lang.c++. Re: Sorting records using sort()

> >>  
> >> *The trouble with writing iterators for this problem is that he needs it*  
> >> *to be variable size at run time and that the iterators have to be*  
> >> *dereferenceable and the dereferenced types assignable. Add his*  
> >> *requirement to not allocate even enough memory to hold pointers to each*  
> >> *object, I can't figure out how you could do it.*  
> >  
> > *Thanks Ron, I think that is definitely a problem. But I still "think" it*  
> > *can be done, dont know how to thou...:)*  
>  
> *I also think it can be done. However, it requires so much trickery that it*  
> *is not interesting to do so. Any of the other options quickly become very*  
> *appealing. My approach uses some dynamic memory, but probably less than an*  
> *array of pointers would take. The actual amount depends on the number of*  
> *copies std::sort makes internally, there is no way around this. I guess*  
> *for a typical implementation this would be around log(n) copies, unless*  
> *the implementation takes special care to dispose of temporaries. You*  
> *cannot count on any of this though.*  
>

Hopefully, by the end of this thread, we'll have something simple  
enough  
to implement...:)

> *OK, how can we do it? Obviously every iterator must know the size of the*  
> *object it is supposed to handle. The main problems we are facing:*  
>  
> - *We cannot create real objects (directly), the size of the real object is*  
> *only known at runtime.*  
> - *std::sort is allowed to (and most frequently does) make extra copies of*  
> *objects.*  
>  
> *Assumptions:*  
> - *There are functions to copy these objects and to compare these objects.*  
> - *The objects to be sorted don't need special construction or destruction*  
> *and can be copied by the previously mentioned routine. They can be*  
> *constructed by copying into a new memory area.*  
> - *std::sort never uses pointers to objects, only iterators. I think the*  
> *standard mandates this, but couldn't find it.*  
>

Would be interesting to know, what std::sort() is required to have  
from  
the standard. I think you are perfectly right that sort doesnt use  
pointers  
to the objects, only iterators and derefernces(\*) the iterators...

> *We create a proxy class and let the iterator return proxy objects. These*  
> *proxy objects store a pointer to some master descriptor that stores the*  
> *location and size of the array and the size of the individual objects. We*  
> *could store this in the proxy object itself, but that seems wasteful.*

## comp.lang.c++. Re: Sorting records using sort()

- > Also, the proxy object stores a void\* to store the data for the real
- > object.
- >
- > This proxy class is what our iterators value\_type is.
- >
- > Now the proxy object implements some intelligent constructors. I doubt we
- > need a default constructor, the value type needs to be assignable only
- > (this follows from the iterator requirements). So we create a constructor
- > that is used only inside the iterator, it sets the data pointer to point
- > inside the array at the correct location. If the copy constructor is
- > called (which would be by std::sort) we dynamically allocate memory to
- > hold the object and use the copying routine to fill it.
- >
- > The assignment operator must use the copying routine if the destination is
- > inside the array. If not, it can either use copy-construct-and-swap or the
- > copying routine.
- >
- > The destructor would check if the pointer points inside the array and if
- > not deletes the object.
- >
- > Obviously, we need to define an operator< for the proxy objects, this
- > calls the comparison function above. If we template the proxy objects, we
- > can use a functor for the comparison, potentially inlining the comparison
- > (this would be the only reason to use std::sort anyhow, so it makes
- > sense).
- >
- > Looking at the standard, this would fulfill all requirements for
- > std::sort I could find. Somehow I think I did overlook something here, so
- > scrutiny by others is appreciated. I also left obvious details out,
- > obviously :-)
- >
- > A note on exceptions. This would give the basic guarantee, if the copying
- > and comparison routines give at least the basic guarantee.
- >
- > A note on memory usage. Assuming that std::sort holds  $\log(n)$  copies of
- > objects internally, this approach uses less memory than the
- > sort-array-of-pointers approach if  $\log(n)*m < n*\text{sizeof}(\text{void}^*)$ . So this
- > greatly depends on these variables, but quickly holds for large  $n$  and gets
- > less interesting for large  $m$ . Assume  $\text{sizeof}(\text{void}^*)=4$ , we get  $m <$
- >  $4n/\log(n)$ . For  $n=100$ ,  $m$  should be less than 60. For  $n=1000$ ,  $m$  should be
- > less than 400. These are gross approximations but should give you a feel.
- >

Usually this is certainly the case for my data.

- > I hope anyone sees that you must have good reasons to do this. It is ugly,
- > errorprone, prone to assumptions your implementation of std::sort makes
- > (in this implementation &\*it does not return a pointer to the real object,
- > any pointer arithmetic will fail), difficult to maintain. There is no real
- > reason why you should not call qsort in the first place, except as an
- > academic exercise.

Re: Sorting records using sort()

comp.lang.c++. Re: Sorting records using sort()

>

First problem to qsort: It's slower than using sort()  
Second problem to qsort: Later I might want to use other STL  
algorithms on this data!

like for\_each()? find()? ...?

Thanks a lot for your comments,  
--Elijah

----

```
[ comp.std.c++ is moderated.  To submit articles, try just posting with ]  
[ your news-reader.  If that fails, use mailto:std-c++@ncar.ucar.edu    ]  
[           --- Please see the FAQ before posting. ---                 ]  
[ FAQ: http://www.jamesd.demon.co.uk/csc/faq.html ]
```