Subject: Re: Bizarre slowness from sort()
Posted by Jonathan Joseph on Tue, 23 Apr 2002 18:56:59 GMT
View Forum Message <> Reply to Message

I had the following enlightening discussion about qsort with a
programmer from RSI who responded to my post, but I'm still not
entirely convinced. Can anyone comment on this?

-Jonathan

(posted with permission)

RSI

You've stumbled over a well known flaw in the standard qsort()
function, which forms the heart of IDL's SORT. Jon Bentley (the
most excellent Programming Pearls guy) wrote a very good
column about this for the now defunct Unix Review about a decade ago:

149.J. L. Bentley, Software Exploratorium: The Trouble With
Qsort, UNIX Review, Vol. 10, 2, pp. 85--93, February 1992.

I am not able to locate a copy for you online, so I am recalling this
from memory:

- qsort(), as implemented in Unix V7, would go quadratic
  in performance when presented with input that is already
  in approximate sorted order. Ironically, input that is not
  close to sorted order sorts very quickly,

- qsort(), as found in almost all operating systems today, is
  based on that now very ancient V7 code.

- Bentley recommended that vendors should modify their qsort()
  implementations to perturb the input order so as to duck this
  aspect of the straightforward quicksort algorithm. Naturally,
  this is a significant complication to an otherwise simple
  implementation.

It would appear that Sun and Microsoft have not taken this advice, but
that HP has. Your example of a large array with only a few distinct
values would be a pathological case in this context.

It should be clear why your experiment with adding the random offset
helped: Your input is no longer essentially pre-sorted, and it no longer
is made up of just a few distinct values.

ME
--

Thank you,

You say that you are recalling this from memory, but the description
of the problem as you phrase it (slow when list is approximately
sorted) does not necessarily match the cases in which I am seeing the
problem. The slowness seems mostly due to the list having only a few
distinct values, whether or not it is nearly sorted. If I make a list
with just a few values that are completely randomly distributed, it
still takes a very long time (at least on SUN and WIN2K running IDL
5.5).
So, perturbing the input order does not speed up the sort.

Example:

```
IDL> a = long(randomu(seed,400000) * 10)
IDL> b = sort(a)
```

RSI

You're welcome. I just wish I had a pointer to the actual article for
you...

> You say that you are recalling this from memory...

Yes, from memory, and 10 year old memory at that. Don't put too much
faith in it. You did mention a case in which you perturbed the input
with a small random number and that the sort did speed up. This does
match the Bentley paper...

Now that you raise the point, I have to agree that your results with
a large array containing only a few distinct values is different than
the case I was describing, but I think it is plausible to think that
there
might be a connection, or that it is a different qsort() pathology of
the
same type.

The best thing you could do, if you had a service contract with Sun,
would be to file a bug. Microsoft too, though I am less confident that
they will do anything about it. Getting the vendor to fix this would be
a great thing for anyone who uses that system.

Here is one fact that I am 100% sure of: IDL uses qsort(), and does not
try to solve the sorting problem itself. It sounds tempting, but I'm
sure
you can see the potential pitfalls. That's why we (presumably) have
standard
libraries for such things...

...if you do find some evidence that it is IDL, and not qsort(), then
I would definitely be interested in hunting that down. I can't see it
though, as IDL's sort is a pretty think wrapper on top of qsort().