

Re: Comparing lists

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2005-10/msg01936.html>

- *From:* "Christian Stapfer" <nil@xxxxxxx>
 - *Date:* Sun, 16 Oct 2005 07:47:17 +0200
-

"Steven D'Aprano" <steve@xxxxxxxxxxxxxxxxxxxxxxxx> wrote in message
news:pan.2005.10.15.19.22.50.864680@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

> On Sat, 15 Oct 2005 18:17:36 +0200, Christian Stapfer wrote:

>

>>>> I'd prefer a (however) rough characterization
>>>> of computational complexity in terms of Big-Oh
>>>> (or Big-whatever) *anytime* to marketing-type
>>>> characterizations like this one...

>>>

>>> Oh how naive.

>>

>> Why is it that even computer science undergrads
>> are required to learn the basics of Big-Oh and
>> all that?

>

> So that they know how to correctly interpret what Big O notation means,
> instead of misinterpreting it. Big O notation doesn't tell you everything
> you need to know to predict the behaviour of an algorithm.

Well, that's right. I couldn't agree more:
it doesn't tell you *everything* but it does
tell you *something*. And that *something*
is good to have.

> It doesn't even tell you most of what you need to know about its
> behaviour.
> Only actual *measurement* will tell you what you need to know.

Well, that's where you err: Testing doesn't
tell you everything *either*. You need *both*:
a reasonable theory *and* experimental data...
If theory and experimental data disagree,
we would want to take a closer look on both,
the theory (which may be mistaken or inadequate)
and the experiment (which may be inadequate or
just plain botched as well).

> Perhaps you should actually sit down and look at the assumptions,
> simplifications, short-cuts and trade-offs that computer scientists make

Re: Comparing lists

- > when they estimate an algorithm's Big O behaviour. It might shock you out
- > of your faith that Big O is the be all and end all of algorithm planning.
- >
- > For all but the simplest algorithm, it is impractical to actually count
- > all the operations — and even if you did, the knowledge wouldn't help
- > you, because you don't know how long each operation takes to get executed.
- > That is platform specific.
- >
- > So you simplify. You pretend that paging never happens. That memory
- > allocations take zero time. That set up and removal of data structures
- > take insignificant time. That if there is an N^2 term, it always swamps
- > an N term. You assume that code performance is independent of the CPUs.
- > You assume that some operations (e.g. comparisons) take no time, and
- > others (e.g. moving data) are expensive.
- >
- > Those assumptions sometimes are wildly wrong. I've been seriously bitten
- > following text book algorithms written for C and Pascal: they assume that
- > comparisons are cheap and swapping elements are expensive. But in Python,
- > swapping elements is cheap and comparisons are expensive, because of all
- > the heavy object-oriented machinery used. Your classic text book algorithm
- > is not guaranteed to survive contact with the real world: you have to try
- > it and see.

Still: the expensiveness of those operations (such as swapping elements vs. comparisons) will only affect the constant of proportionality, not the asymptotic behavior of the algorithm. Sooner or later the part of your program that has the worst asymptotic behavior will determine speed (or memory requirements) of your program.

- > Given all the assumptions, it is a wonder that Big O
- > estimates are ever useful, not that they sometimes
- > are misleading.
- >
- > [snip]
- >>> The marketing department says: "It's $O(N)$, so it is blindingly fast."
- >>>
- >> I might as well interpret "blindingly fast" as meaning $O(1)$. – Why not?
- >> Surely marketing might also have reasoned like
- >> this: "It's $O(1)$, so its blindingly fast". But I **want**, nay, I **must**
- >> know whether it is $O(N)$ or $O(1)$.
- >
- > You might *_want_*, but you don't *_need_* to know which it is, not in every
- > case. In general, there are many circumstances where it makes no
- > sense to worry about Big O behaviour. What's your expected data look like?
- > If your data never gets above $N=2$, then who cares whether it is $O(1)=1$,
- > $O(N)=2$, $O(N^2)=4$ or $O(2^N)=2$? They are all about as fast.
- >
- > Even bubble sort will sort a three element list fast enough — and
- > probably faster than more complicated sorts. Why spend all the time

Re: Comparing lists

> setting up the machinery for a merge sort for three elements?

Since you have relapsed into a fit of mere polemics, I assume to have made my point as regards marketing type characterizations of algorithms ("blazingly fast") vs. measures, however rough, of asymptotic complexity measures, like Big-Oh. – Which really was the root of this sub-thread that went like this:

...>> To take the heat out of the discussion:

... >> sets are blazingly fast.

... > I'd prefer a (however) rough characterization
... > of computational complexity in terms of Big-Oh
... > (or Big-whatever) *anytime* to marketing-type
... > characterizations like this one...

> [snip]

>

>>> Big O notation is practically useless for judging how fast a single
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

>>> algorithm will be, or how one algorithm compares to another.

>>

>> That's why Knuth liked it so much?

>> That's why Aho, Hopcroft and Ullman liked it so much? That's why Gonnet

>> and Baeza-Yates liked it so much?

>

> Two reasons: it is useful for telling you how a single algorithm will
> scale as the input increases, just as I said.

Curiously, just a few lines before writing this, you have polemically denied any "practical" use for Big-Oh notation.

> And, unlike more accurate ways of calculating the speed of an algorithm
> from first principles, it is actually possible to do Big O calculations.

Right. It's a compromise: being somewhat precise – without getting bogged down trying to solve major combinatorial research problems...

> No doubt the state of the art of algorithm measurements has advanced since
> I was an undergraduate, but certain fundamental facts remain: in order to
> calculate that Big O, you have to close your eyes to all the realities
> of practical code execution, and only consider an idealised calculation.

That's right. Nothing stops you from then opening your eyes and testing some code, of course. *But* always try to relate what you see there with what theoretical grasp of the situation you have. If experimental data and theory *disagree*: try to

Re: Comparing lists

fix the experiment and/or the theory.

- > Even when your calculation gives you constants of proportionality and
- > other coefficients, Big O notation demands you throw that information
- > away.
- >
- > But by doing so, you lose valuable information. An $O(N^2)$ algorithm that
- > scales like $1e-6 * N^2$ will be faster than an $O(N)$ algorithm that scales
- > as $1e6 * N$, until N reaches one million million. By tossing away those
- > coefficients, you wrongly expect the first algorithm to be slower than the
- > second, and choose the wrong algorithm.
- >
- >>> It is only useful for telling you how a single algorithm will scale as
- >>> the input increases.
- >>
- >> And that's really very useful information indeed.
- >
- > Yes it is. Did I ever say it wasn't?

Well yes, by the way you attacked Big-Oh notation as "practically useless" (see above) I assumed you did.

- >> Since, given such
- >> information for the basic data types and operations, as implemented by
- >> the language and its standard libraries, I stand a real chance of being
- >> able to determine the computational complexity of the
- >> *particular*combination* of data types and algorithms of my own small
- >> utility or of a critical piece of my wonderful and large application, on
- >> which the future of my company depends, with some confidence and
- >> accuracy.
- >
- > Yes, zero is a real chance.
- >
- >
- > [snip]
- >
- >>> As for sets, they are based on dicts, which are effectively hash
- >>> tables. Hash tables are $O(1)$, unless there are collisions,
- >>
- >> Depending on the "load factor" of the hash tables. So we would want to
- >> ask, if we have very large lists indeed, how much space needs to be
- >> invested to keep the load factor so low that we can say that the
- >> membership test is $O(1)$.
- >
- > And knowing that hash tables are $O(1)$ will not tell you that, will it?
- >
- > There is only one practical way of telling: do the experiment. Keep
- > loading up that hash table until you start getting lots of collisions.
- >
- >> Do A-B and A&B have to walk the entire hash

Re: Comparing lists

Re: Comparing lists

>> table (which must be larger than the sets, because of a load factor <
>> 1)? Also: the conversion of lists to sets needs the insertion of N
>> elements into those hash tables. That alone already makes the overall
>> algorithm *at least* O(N). So forget about O(log N).
>
> Yes, inserting N items into a hash table takes at least N inserts. But if
> those inserts are fast enough, you won't even notice the time it takes to
> do it, compared to the rest of your algorithm. In many algorithms, you
> don't even care about the time it takes to put items in your hash table,
> because that isn't part of the problem you are trying to solve.
>
> So in real, practical sense, it may be that your algorithm gets dominated
> by the O(log N) term even though there is technically an O(N) term in
> there. Are Python dicts like that? I have no idea. But I know how to find
> out: choose a problem domain I care about ("dicts with less than one
> million items") and do the experiment.
>
>
>>> in which case the more
>>> common algorithms degenerate to O(N).
>>>
>> So here, indeed, we have the kind of reasoning that one ought to be able
>> to deliver, based on what's in the Python documentation. Luckily, you
>> have that kind the knowledge of both, how sets are implemented and what
>> Big-Oh attaches to the hash table operation of "look up".
>> In order to *enable* SUCH reasoning for *everyone*,
>> starting from the module interface documentation only, one clearly needs
>> something along the lines that I was suggesting...
>
> I don't object to having that Big O information available, except
> insofar as it can be misleading, but I take issue with your position that
> such information is necessary.

Blindly testing, that is, testing *without* being
able to *relate* the outcomes of those tests (even
the *design* of those tests) to some suitably
simplified but not at all completely nonsensical
theory (via Big-Oh notation, for example), is *not*
really good enough.

>>> Now run the test code:
>>>
>>> py> test_Order()
>>> N = 1 0.000219106674194
>>> N = 10 0.000135183334351
>>>
>> Curious: N=10 takes less time than N=1?
>
> Yes, funny how real-world results aren't as clean and neat as they are in
> theory. There are those awkward assumptions coming to bite you again. I've
> done two additional tests, and get:

Re: Comparing lists

```
>
> N = 1 0.000085043907166
> N = 10 0.000106656551361
>
> N = 1 0.000497949123383
> N = 10 0.000124049186707
>
> Remember, these results are averaged over twenty trials. So why it is
> quicker to do work with sets of size 10 than sets of size 1? Big O
> notation will never tell you, because it ignores the implementation
> details that really make a difference.
>
>
>
>>> N = 100 0.000481128692627
>>
>> Why do we have such a comparatively large jump here, from N=100 to
>> N=1000? Did hash tables overflow during conversion or something like
>> that?
>
> Who knows? Maybe Python was doing some garbage collection the first time I
> run it. I've modified my code to print a scale factor, and here is another
> run:
>
> N = 1 0.00113509893417
> N = 10 0.000106143951416 (x 0.093511)
> N = 100 0.00265134572983 (x 24.978774)
> N = 1000 0.0057701587677 (x 2.176313)
> N = 10000 0.0551437973976 (x 9.556721)
> N = 100000 0.668345856667 (x 12.120055)
> N = 1000000 8.6285964489 (x 12.910376)
>
> An increase from N=1 to 1000000 (that's a factor of one million) leads to
> an increase in execution time of about 7600.
>
> You will notice that the individual numbers vary significantly from trial
> to trial, but the over-all pattern is surprisingly consistent.
>
>
>>> N = 1000 0.0173740386963
>>> N = 10000 0.103679180145
>>> N = 100000 0.655336141586
>>> N = 1000000 8.12827801704
>>
>> Doesn't look quite O(n). Not yet...
>
> No it doesn't.
>
>>
>>> In my humble opinion, that's not bad behaviour. It looks O(log N) to
>>> me,
```

Re: Comparing lists

Re: Comparing lists

>
> That's a mistake — it is nowhere near $O(\log N)$. My bad. Closer to
> $O(\sqrt{N})$, if anything.
>
>
>> How could that be? *Every* element of A and B must be touched, if only to
>> be copied: that can't make it $O(\log(N))$.
>
> And, no doubt, if you had *really enormous* lists, oh, I don't know, maybe
> a trillion items, you would see that $O(N)$ behaviour. But until then, the
> overall performance is dominated by the smaller-order terms with larger
> coefficients.
>
>
>> Also, the conversion of lists
>> to sets must be at least $O(N)$. And N isn't the right measure anyway. It
>> would probably have to be in terms of $|A|$ and $|B|$. For example, if $|A|$
>> is very small, as compared to $|B|$, then $A-B$ and $A \cap B$ can be determined
>> rather quickly by only considering elements of A.
>
>
> Both lists have the same number of elements, so double N.
>
>
> [snip]
>
>> You must distinguish questions of principle and questions of muddling
>> through like this testing bit you've done.
>
> Your "question of principle" gives you completely misleading answers.
> Remember, you were the one who predicted that lists would have to be
> faster than sets.

I didn't say they would *have* to be faster
– I was mainly asking for some *reasoned*
argument why (and in what sense) conversion
to sets would be an "efficient" solution
of the OPs problem.

> Your prediction failed miserably.

Interestingly, you admit that you did not
really compare the two approaches that
were under discussion. So your experiment
does *not* (yet) prove what you claim it
proves.

>> It would take me some time to
>> even be *sure* how to interpret the result.
>
> What's to interpret? I know exactly how fast the function will run, on

Re: Comparing lists

Re: Comparing lists

- > average, on my hardware. I can even extrapolate to larger sizes of N,
- > although I would be very careful to not extrapolate too far. (I predict
- > less than 10 minutes to work on a pair of 10,000,000 element lists, and
- > less than two hours to work on 100,000,000 element lists.)

For a starter: You have chosen a very particular type of element of those lists / sets: integers. So the complexity of comparisons for the OPs application might get **seriously** underestimated.

- >
- >> I would never want to say
- >> "it looks $O(\log N)$ to me", as you do, and leave it at that. Rather, I
- >> might say, as you do, "it looks $O(\log N)$ to me", **but** then try to
- >> figure out, given my knowledge of the implementation (performance wise,
- >> based on information that is sadly missing in the Python documentation),
- >> **why** that might be.
- >
- > Fine. You have the source code, knock yourself out.

That's just what I do **not** think to be a particularly reasonable approach. Instead, I propose stating some (to the implementer **easily** available) information about asymptotic behavior of operations that are exposed by the module interface upfront.

- >> Then, if my experiments says "it looks like $O(\log$
- >> $N)$ " AND if my basic knowledge of the implementation of set and list
- >> primitives says "it should be $O(\log N)$ " as well, I would venture, with
- >> some **confidence**, to claim: "it actually IS $O(\log N)$ "....
- >>
- >> You do not compare the convert-it-to-sets-approach
- >> to the single list-walk either.
- >
- > No I did not, because I didn't have a function to do it.

Here we see one of the problems of a purely experimentalist approach to computational complexity: you need an implementation (of the algorithm and the test harness) **before** you can get your wonderfully decisive experimental data.

This is why we would like to have a way of (roughly) estimating the reasonableness of the outlines of a program's design in "armchair fashion" – i.e. without having to write any code and/or test harness.

- > You've got my
- > source code. Knock yourself out to use it to test any function you like.
- >
- >> Supposing the OP had actually sorted
- >> lists to begin with, then a single, simultaneous walk of the lists would

Re: Comparing lists

>> be about as fast as it can get. Very, very likely **faster** than
>> conversion to sets would be...
>
> Please let us know how you go with that. It should be really interesting
> to see how well your prediction copes with the real world.
>
> (Hint: another of those awkward little implementation details... how much
> work is being done in C code, and how much in pure Python? Just something
> for you to think about. And remember, an $O(N)$ algorithm in Python will be
> faster than an $O(N^2)$ algorithm in C... or is that slower?)

This discussion begins to sound like the recurring arguments one hears between theoretical and experimental physicists. Experimentalists tend to overrate the importance of experimental data (setting up a useful experiment, how to interpret the experimental data one then gathers, and whether one stands any chance of detecting systematic errors of measurement, all depend on having a good **theory** in the first place). Theoreticians, on the other hand, tend to overrate the importance of the coherence of theories. In truth, **both** are needed: good theories **and** carefully collected experimental data.

Regards,
Christian

—

»When asked how he would have reacted if Eddington's **measurements** had come out differently, Einstein replied: "Then I would have been sorry for him
– the **theory** is correct."«
– Paul B. Armstrong: 'Conflicting Readings'

.

• *Follow-Ups:*

- ◆ **Re: Comparing lists**
 ◇ *From:* Alex Martelli
- ◆ **Re: Comparing lists**
 ◇ *From:* Ron Adam

• *References:*

- ◆ **Re: Comparing lists**
 ◇ *From:* Christian Stapfer
- ◆ **Re: Comparing lists**
 ◇ *From:* Christian Stapfer
- ◆ **Re: Comparing lists**
 ◇ *From:* Scott David Daniels

Re: Comparing lists

- ◆ **Re: Comparing lists**
 - ◇ *From:* jon
- ◆ **Re: Comparing lists**
 - ◇ *From:* Christian Stapfer
- ◆ **Re: Comparing lists**
 - ◇ *From:* Steven D'Aprano
- ◆ **Re: Comparing lists**
 - ◇ *From:* Christian Stapfer
- ◆ **Re: Comparing lists**
 - ◇ *From:* Steven D'Aprano

- Prev by Date: **Re: How to get a raised exception from other thread**
- Next by Date: **Re: Some set operators**
- Previous by thread: **Re: Comparing lists**
- Next by thread: **Re: Comparing lists**
- Index(es):
 - ◆ **Date**
 - ◆ **Thread**