Ruby master - Feature #1089
Stable sorting for sort and sort_by
02/02/2009 04:32 PM - murphy (Kornelius Kalnbach)

Status: Rejected
Priority: Normal
Assignee: matz (Yukihiro Matsumoto)
Target version:

Description

I'd like to have stable sorting in Ruby. Stable sorting means that if two items are ==, they will retain their order in the list after sorting. In other words, stable sort does never switch two equal items.

In Ruby 1.9.1, you'd have to use something like

enum.sort_by.with_index { |a, i| [a, i] }

to achieve a stable sort in Ruby.

It would also be nice to minimize the number of comparisons, since method calling is expensive in Ruby.

Python is using a highly optimized implementation of a variant of merge sort, which is stable and uses very few comparisons (called Timsort after Tim Peters); maybe Ruby can adopt it for sort and sort_by:

http://svn.python.org/projects/python/trunk/Objects/listsort.txt

Introducing stable sorting would not be a problem for old code because the order of equal items after sorting is currently not specified.

History

#1 - 02/02/2009 11:36 PM - headius (Charles Nutter)

Kornelius Kalnbach wrote:

I'd like to have stable sorting in Ruby. Stable sorting means that if two items are ==, they will retain their order in the list after sorting. In other words, stable sort does never switch two equal items.

... It would also be nice to minimize the number of comparisons, since method calling is expensive in Ruby.

Python is using a highly optimized implementation of a variant of merge sort, which is stable and uses very few comparisons (called Timsort after Tim Peters); maybe Ruby can adopt it for sort and sort_by:

http://svn.python.org/projects/python/trunk/Objects/listsort.txt

Introducing stable sorting would not be a problem for old code because the order of equal items after sorting is currently not specified.

#2 - 02/03/2009 10:13 AM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Feedback
- Assignee set to matz (Yukihiro Matsumoto)

#3 - 02/06/2009 03:06 AM - murphy (Kornelius Kalnbach)
Hello,

it took me some time to answer. Your question wasn't easy ;)

Charles Oliver Nutter wrote:

I'd love to hear why you're interested in a stable sort.

Two use cases come to my mind where stable sort is needed. Excuse me for the confusing examples, but I find it always hard to talk about this topic.

The first one is a bit pathetic. A list of different objects, some of which are ==, but the list is already sorted. Unstable sort might change the list order, which might surprise some people.

```ruby
# Something that can be == without being the same.
class StrangeInt < Struct.new :value
  def <=> other
    self.value / 2 <=> other.value / 2
  end
  include Comparable
  def inspect
    value
  end
end
StrangeInt.new(4) == StrangeInt.new(5)  # => true

a = Array.new(10) {StrangeInt.new(i)}
a # => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# stable sort would return this
a.sort_by { |x| [x.value / 2, x.value] }
# => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

a.sort
# => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

a.sort_by { |x| x }
# => [0, 1, 3, 2, 4, 5, 6, 7, 8, 9]

(This also shows that currently, Array#sort does not check if the list is already sorted.)
```

The second example is sort_by chaining. Imagine you have a table view of files with name, file type, and file size. When you're searching for the biggest PDF file, you first click twice on the "size" column and then once on the "type" column, and scroll down to the .pdf files to find the biggest one on the top. A Rails-style variant:

```ruby
files.sort_by(&:size).reverse.sort_by(&:type)
```

...which doesn't work as expected with unstable sort! You need to

```ruby
files.sort_by { |file| [file.type, -file.size] }
```

I think stable sort feels more natural here.

And the -file.size trick doesn't even work for strings! We had such a case in a project, and in the end we couldn't implement it the way we wanted. With a stable sort, our implementation would become much more clean. I think the example is a very strong case for stable sorting.

In JRuby, we recently put in place an unstable hybrid of a couple sorts so we'd have competitive performance. Our original sort was exactly what you talk about for Python, a "pretty fast" stable modified merge sort.

Oh noes ^_^ How much difference did it make?

I'd also be interested to see how the performance of Timsort compares to what's currently in Ruby and JRuby as well as compared to the built-in JDK sort we used before. If stable sorting is desired, we could certainly provide a flag or flip a bit in JRuby to provide it.
I don't know how timsort would perform compared to the current MRI quicksort or JRuby's sort implementation. I assume it takes some time to port it to Ruby for testing, and my experience with Ruby core programming is limited. But I think the question of a need for stable sorting is independent from that.

[murphy]

#4 - 02/06/2009 11:38 PM - headius (Charles Nutter)
Kornelius Kalnbach wrote:

Charles Oliver Nutter wrote:

I'd love to hear why you're interested in a stable sort.

Two use cases come to my mind where stable sort is needed. Excuse me for the confusing examples, but I find it always hard to talk about this topic.

Your two cases make sense to me, and are similar to reasons I gave for us not to abandon a stable sort. I eventually overruled myself, however, after more published benchmark results put us slower than Ruby 1.8 for array sorts. Perception is important; more important than sort stability, I have not decided ;)

In JRuby, we recently put in place an unstable hybrid of a couple sorts so we'd have competitive performance. Our original sort was exactly what you talk about for Python, a "pretty fast" stable modified merge sort.

Oh noes ^_^ How much difference did it make?

I guess we ended up going with an unstable sort because it did make a difference; I think keeping the sort stable makes it harder to combine algorithms for different classes of list (partially sorted, reversed, randomized) which seemed to be key to matching MRI's sort performance. I raised a concern about stability, but we figured if nobody had seen issues in MRI nobody would care if JRuby's sort became unstable.

Perhaps this is a case where having the fast unstable sort support a slower, possibly ruby-based, stable sort is the way to go. In general a ruby-based sort performs pretty well on both JRuby and Ruby 1.9. Here's some recent results:

http://blog.concord.org/archives/24-Comparison-of-Ruby-1.8.6-1.9-and-JRuby-running-on-Java-1.5-1.6-and-1.7.html

It certainly wouldn't be as fast as a native sort, but perhaps that's not a big deal?

* Charlie

#5 - 02/07/2009 12:57 AM - murphy (Kornelius Kalnbach)
Charles Oliver Nutter wrote:

Your two cases make sense to me, and are similar to reasons I gave for us not to abandon a stable sort. I eventually overruled myself, however, after more published benchmark results put us slower than Ruby 1.8 for array sorts. Perception is important; more important than sort stability, I have not decided ;)

actually, I'd prefer the feature of "stable sort" over "fast sort". I use Ruby for convenience, not for speed. still: what is the tradeoff?

I guess we ended up going with an unstable sort because it did make a difference; I think keeping the sort stable makes it harder to combine algorithms for different classes of list (partially sorted, reversed, randomized) which seemed to be key to matching MRI's sort performance. I raised a concern about stability, but we figured if
nobody had seen issues in MRI nobody would care if JRuby's sort became unstable.

it's critical how much the difference was...if we get stability for, say, 15% more sort time, I'd buy it. if even the best algorithms are 50x slower, I'd abandon my request. but I don't think this is the case; more investigation follows.

anything that performs better than this would be appreciated (stable sort is roughly 50 times slower):

time ruby19 -e 'Array(1..1000000).shuffle.sort'
real 0m0.529s
real 0m0.164s (without sorting)

i = 0; Array(1..1000000).shuffle.sort_by { |o| [o, i += 1] }
real 0m18.742s

Array(1..1000000).shuffle.sort_by.with_index { |o, i| [o, i] }
real 0m18.430s

Array(1..1000000).shuffle.sort_by.with_index { |*a| a }
real 0m18.383s

Perhaps this is a case where having the fast unstable sort support a slower, possibly ruby-based, stable sort is the way to go. In general a ruby-based sort performs pretty well on both JRuby and Ruby 1.9. Here's some recent results:

http://rubyurl.com/3d87

It certainly wouldn't be as fast as a native sort, but perhaps that's not a big deal?

it is...using their benchmark mergesort:

-rmergesort -e 'mergesort Array(1..1000000).shuffle'
real 0m18.066s

wow, that's faster than the sort_by approach :D my own in-place merge sort version (based on the one Java uses internally for Object[]) is even faster:

-rmerge_sort -e 'Array(1..1000000).shuffle.merge_sort!'
real 0m10.401s

so, the best I currently get with pure Ruby is about 30x. I'm sure we can do better with a Java or C implementation.

let's see timsort now. some Python 2.5 benchmarks:

time python2.5 -c 'from random import shuffle; list = range(1000000); shuffle(list); sorted(list)'
real 0m2.432s
real 0m1.191s (without sorting)

so I guess timsort takes about 1.3s, which would be about 3.5x slower than Ruby's sort. this is not very nice; but 1.3s are still much better than 10s!

so, maybe stable sort should be optional. what options do we have?

* compiler switch, global variable and such; we don't want that
* a new method, like Enumerable#stable_sort. my code example would become:

files.stable_sort_by(&:size).reverse.stable_sort_by(&:type)

* an optional parameter to sort and sort_by:

enum.sort(stable=false) -> array
I know boolean parameters are not nice, but we already use them in Object#methods. My code example would become:

```ruby
files.sort_by(true, &:size).reverse.sort_by(true, &:type)
```

None of these options feels like Ruby to me. I hope somebody has another idea.

[murphy]

#6 - 02/07/2009 01:06 AM - murphy (Kornelius Kainbach)

Charles Oliver Nutter wrote:

In JRuby, we recently put in place an unstable hybrid of a couple sorts so we'd have competitive performance.

Can you remember which version of JRuby still contains the stable sort? I'd like to do benchmarks...

[murphy]

#7 - 09/19/2009 05:58 AM - marcandre (Marc-Andre Lafortune)

- Status changed from Feedback to Open

#8 - 10/11/2009 01:42 AM - naruse (Yui NARUSE)

- Status changed from Open to Rejected

#9 - 11/06/2009 12:00 PM - murphy (Kornelius Kainbach)

I'm not sure why the proposal was rejected. Can somebody explain?

If we don't want to change the #sort implementation for performance reasons, I can open another ticket for the request of an alternative, fast and stable sort method in core Ruby. I still think that Ruby should have a core method for stable sorting.

#10 - 11/06/2009 04:14 PM - matz (Yukihiro Matsumoto)

Hi,

In message "Re: [ruby-core:26557] [Feature #1089] Stable sorting for sort and sort_by" on Fri, 6 Nov 2009 12:00:35 +0900, Kornelius Kainbach redmine@ruby-lang.org writes:

I'm not sure why the proposal was rejected. Can somebody explain?

As far as I know, sort stability is rarely required, not worthy for slows down sorting in general. If there's stable sort routine faster than current one, please tell me.

If we don't want to change the #sort implementation for performance reasons, I can open another ticket for the request of an alternative, fast and stable sort method in core Ruby. I still think that Ruby should have a core method for stable sorting.

If you really need stable sort, you can do so using sort_by pretty easily.

```ruby
i = 0
ary.sort_by{|x| [x, i += 1] }
```

[matz]

#11 - 11/06/2009 04:23 PM - naruse (Yui NARUSE)

Because you can't persuade Charles (and matz).
I maintain tickets which doesn't concluded for long time. Rejecting tickets kills or reactivates it.

You can open another ticket for an alternative or reopen this with new implementation or use cases which strongly needs stable sorting. Seeing above matz' comment, you should show use cases which needs fast and stable sort.

#12 - 05/02/2019 04:56 PM - jonathanhefner (Jonathan Hefner)

Description updated

One use case I have for a stable sort is a large CSV file which is tracked in a git repository. I have a script which loads the file, adds rows, and then sorts all rows by a particular column before writing them back to the file. I need to minimize the git diff for the file, so I use a stable sort. I currently use the sort_by.with_index technique, but it would be nice to have faster performance. (A benchmark currently shows sort_by.with_index is 7x slower than just sort_by.)

Another use case I have is a list of tasks with optional priority. By default, the tasks are ordered manually with no priority set. But when a priority is set, the tasks are sorted such that higher priority tasks come first, while still preserving the manual ordering of both the prioritized and unprioritized tasks.

In addition to performance benefits, I think something like sort_by(stable: true, &:foo) communicates purpose more clearly than sort_by.with_index([{x, i[x,foo, i]}].

#13 - 11/07/2019 03:58 AM - rogerdpack (Roger Pack)

It's sad that #sort on linux is "mostly stable" but on OS X is unstable [1]. It's confusing, unfortunately.

Since almost always when I do a "sort_by(&x).sort_by(&y)" I am looking for a stable sort. And that's exactly the reason that java gives for defaulting to a stable sort if it's dealing with objects [2]. Might be nice to add a stable sort option, here's a fork with a timsort:

https://github.com/ruby/ruby/compare/master...phluid61:timsort

benchmarks are here: https://www.ruby-forum.com/t/timsort-in-ruby/235311/7 It's slightly slower than the unstable sort.

Anyway it would be nice to at least have it around in core, and, in my mind, be the default for at least sort_by but that's up to discretion...

[1] https://github.com/crystal-lang/crystal/issues/2350#issuecomment-550188746
[2] https://stackoverflow.com/a/15154287/32453 "Stability is a big deal when sorting arbitrary objects. For example, suppose you have objects representing email messages, and you sort them first by date, then by sender. You expect them to be sorted by date within each sender, but that will only be true if the sort is stable."

#14 - 11/07/2019 04:29 AM - sawa (Tsuyoshi Sawada)

geroderpack (Roger Pack) wrote:

It's sad that #sort on linux is "mostly stable" but on OS X is unstable [1]. It's confusing, unfortunately.

Do you really mean it's sad? Or do you mean it's said?

#15 - 11/07/2019 04:00 PM - shevegen (Robert A. Heiler)

sawa wrote:

Do you really mean it's sad? Or do you mean it's said?

I think he meant "said", rather than "said", possibly due to the desire (by a ruby user) to see that ruby behaves in the same way across different operating systems - at the least this is what I think rogerdpack meant. :) 

I myself have no particular pro/con opinion on the issue; it may be better to create a new one, in the event that people feel strongly about it (11 years ago is quite a long time). I can, however had, understand that ruby users would like to see ruby behaves in the same way whenever they use it. In that sense, ruby abstracts away complexities and oddities of different platforms - that is convenient for a user.

I think matz and the core team also have that goal, in the sense that I remember suggestions that were rejected in the past it would have led to different behaviour on different operating systems (or disparate features, e. g. things that would only work on linux+ruby, but not on windows+ruby or mac+ruby).

To sorting in general: some time ago the insert-order remained the same for hashes, so that "first-in" or "last-in" could be distinguished for free. I liked that addition. Whenever I use .each_pair on a hash, I know that the more recent additions will, by default, appear at the "end" of the output, if we use e. g. .each_pair combined with puts/print output. I think that was a good change - we sort of gain additional information for free, which can be helpful sometimes. So I think that was convenient.
On the particular issue here, I have no particular opinion, but I also think that both Murphy and Rogerdpack made good points. Even then I think it is better to make a new proposal, as Naruse suggested back then (sorry for commenting on that old issue as well). Matz objection was very much in regards to speed/slow-downs, so I think new proposals should consider any speed trade-off as well, if it may affect other Ruby users and their code.

#16 - 03/25/2020 04:58 AM - Rogerdpack (Roger Pack)

"sad"

It's just surprising that "depending on OS" sometimes sort is always stable. Then occasionally isn't. Without it nested sort_by's don't work as you'd expect (secondary, tertiary sort), without high performance hit of the #with_index trick (and high cognitive hit LOL). Java uses stable sort for objects for just this reason, FWIW. I don't quite feel qualified to do a new ticket :| ... Cheers!