









It's not a bug in sort - it's how <=> works for everything except floats.

== Argument from mathematics ==

Ruby has a few equality-like relations - ==, eql?, equal?.

They differ, but they're all (almost) mathematical equivalence relations.

For all three, these can be expected:

- `a == a`
- `a == b` iff `b == a`
- if `a == b` and `b == c` then `a == c`

I can think of one good exception to `a == a` - with things like float NaNs and sql nulls - but these really are more techniques for handling errors than real values.

I really cannot think of a single case where it would make sense to make equality either non-symmetric, or non-transitive.

Similar reasoning applies to <=> - normally it defines partial order between objects:

- `a >= b` and `b >= a` only if `a == b`
- if `a >= b` and `b >= c` then `a >= c`

Transitivity of equality, and transitivity of partial ordering seem to be violated only in one case in Ruby - for comparisons between floats and other numeric types. (and this causes sort to break).

Can you think of any other type that does something like that?

== Argument from rationals ==

When you have mixed type operations, and one type is bigger, the most obvious thing to do is simply converting argument of smaller type to bigger type.

For example you can define pretty much every Rational vs Integer operation as:

```
class Rational
  def something(x)
    x = x.to_r if x.is_a?(Integer)
    ...
  end
end
```

And you can do this for Complex operations and non-complex arguments etc.

You never do it the other way around - converting to smaller type. This would be obviously wrong:

```
class Integer
  def something(x)
    x = x.to_i if x.is_a?(Rational)
    ...
  end
end
```

So why, if Rational can represent every floating point value (except nans/infinities/negative zero that is), do rational vs float operations downconvert to float, instead of upconverting to rational?

I can think of no other such case anywhere.

With bignum vs float, neither is strictly wider than other, but both can be represented as rationals.

This doesn't mean I want bignum + float to return rationals, downconversion before returning is perfectly fine.

I just want it to be pretty much equivalent to this:

```
class Integer
  def +(x)
    if x.is_a?(Float) and x.finite?
```

