

# Re: Mathematical formula to compare integer sequence

---

*Source:* <http://sci.tech-archive.net/Archive/sci.math/2006-06/msg03668.html>

---

- *From:* "Tim Peters" <[tim.one@xxxxxxxxxxxxx](mailto:tim.one@xxxxxxxxxxxxx)>
  - *Date:* Thu, 29 Jun 2006 22:17:34 -0400
- 

[Abstract Dissonance, trying to tell whether one sequence is a permutation of another]

...  
What I mean is something that is simple to implement. Doesn't necessarily have to be fast as I'm only using it sequences of small size.

Then you have to define the implementation language first :-). For example, I usually reach for Python when experimenting, and the only approach I'd even consider for small sequences in that language is the obvious ("sorted()" and sequence comparison are built in):

```
def is_permutation(a, b):  
    return sorted(a) == sorted(b)
```

I was going to just sort them then compare element wise as that's pretty easy to implement

Depends on the language, but so long as sorting is built in that gives an "obviously correct" approach. In programming and math, "obviously correct" usually beats "not obviously wrong", and both always beat "WTF?!" ;-)

but I was curious as to if there was a mathematical function that would do the same thing. Similar to what I shown above but that doesn't work ;/  
It might be able to be modified somewhat though

$$\text{Product}(|\#(a_k)*a_k - \#(b_k)*b_j|, j=1..n)$$

where  $\#(a_k)$  returns the number of duplicates of the element  $a_k$  in the sequence. I think that might fix the problem and is sorta like using the multiset idea?

Re: Mathematical formula to compare integer sequence

Yup, if done right. I wouldn't try to make this work, because it's both more obscure and less efficient than the sorting approach.

[Tim Peters]

...  
Do define "better" first. You can, e.g., get simple but absurdly impractical solutions via stuff like:

$$f(a) = \text{product}(p_{a_i}, i=1..n)$$

where  $p_{a_i}$  denotes the  $a_i$ 'th prime. For example,

$$f([1,1,1,2]) = 2*2*2*3 = 24$$

and

$$f([2,1,1,1]) = 3*2*2*2 = 24$$

and

$$f([1,2,1,2]) = 2*3*2*3 = 36$$

and

$$f([1,2,3,4]) = 2*3*5*7 = 210$$

etc. It's easy then to prove that  $f(a) = f(b)$  if and only if  $a$  is a permutation of  $b$ . In effect, this `_is_` building a hash-based multiset representation, but in a spectacularly impractical way.

lol... its not that impractical! ;) I could use a look up table for the primes and it would be quite fast.

Are you sure? You later said you can have up to 20 integers each as large as 20, so the product could get as large as:

$$p_{20}^{20} = 71^{20} = 10596610576391421032662867140133202401$$

Again this depends on the language. There's no bound on integer size in Python, but in most languages you'll have to muck with "big integer" packages if you need more than 64 bits.

Dave Rusin's idea of building a polynomial is also elegant, and doesn't require a table of primes, but in computing  $\text{Product}(x_{a_i})$  the constant term can get as large as  $20^{20} = 10485760000000000000000000$ , and again you're outside the range of 64-bit integers.

Ofcourse I don't really want to have to read in an array of primes to do a simple computation. Its not that it can't be done in a simple way using algorithms in C++ but just that I was wondering if there was a simple mathematical way.

## Re: Mathematical formula to compare integer sequence

You have given a pretty simple one in some sense. It is based on primes though which makes it a tad more complicated. Maybe there is another way that doesn't have some hidden complexities involved (i.e., finding primes) but is based strictly on algebra?

$\text{sum}(1 \ll (5 * a_i))$  would be correct (if an element can appear no more than 31 times) and blazing fast in C++ -- if it had 105-bit integers.

similar to (if it works).

$\text{sum}(\text{Product}(\#(a_k) * a_k - \#(b_k) * b_j, j=1..n), k=1..n)$

but even the # function is sorta "algorithmic" .... (ofcourse one can say that sum and product are too but there seems to be some difference between the two... one involves "searching and comparing" while the other is purely computational).

If you peek under the covers and look at the generated machine code, you'll probably find that your absolute values generate compares and branches too.

I'm not too worried about the time complexity though as I'm more interested in its "algorithmic" simplicity (basically just algebra).

So let's respell  $\text{sum}(1 \ll (5 * a_i))$  as:

$H(a) = \text{sum}(32^{a_i})$

Then so long as each  $a_i$  is a non-negative integer, and no  $a_i$  appears more than 31 times,  $H(a) = H(b)$  iff  $a$  is a permutation of  $b$ .

OK, I have to admit that I find the nested sum/product thingies plain ugly. If you're assuming from the start that  $a$  and  $b$  both have length  $n$ , then it's enough that each element in  $a$  appear exactly the same number of times in  $a$  as in  $b$ . At least let yourself use the Kronecker delta function,

$d(i, j) = 1$  if  $i = j$ , or  $0$  if  $i \neq j$

In C classic,  $d(i, j)$  is implemented simply as " $i == j$ ".

Then the number of times  $a_i$  appears in  $a$  is

$\text{sum}(d(a_i, a_j), j=1..n)$

and the number of times  $a_i$  appears in  $b$  is similarly

Re: Mathematical formula to compare integer sequence

$$\sum_{j=1}^n d(a_i, b_j)$$

so

$$d(\sum_{j=1}^n d(a_i, a_j), \sum_{j=1}^n d(a_i, b_j))$$

is 1 if  $a_i$  appears the same number of times in a and b, and 0 if not, so a is a permutation of b iff

$$\sum_{i=1}^n d(\sum_{j=1}^n d(a_i, a_j), \sum_{j=1}^n d(a_i, b_j)) = n$$

But that's nuts ;-)

.