The Most Notoriously Difficult Double-to-Int Truncation Problem in the Entire World

Here's a simple question: what is 7 / 3 - 1?

That might sound like a question put to rest in third grade, but if you're picking up a language like C or C++ or Java for the first time, the answer might catch you off guard. Rather than 1.333333..., or 1 remainder 1, the answer is 1. Thus, (7/3)*3 isn't 7, either, it's 6 – multiplication and division aren't reliable inverses here, not for integers.

This violation of the principle of least surprise is more sensible than you might initially think; computers often have *really* good performance when dealing with whole numbers, and converting between whole numbers and floating point numbers is a performance hit. To avoid that conversion, we rely on a predictable rule for division that takes two whole numbers and yields another whole number, even if it violates our math class expectations. We do this by truncating the answer – simply discarding the remainder or fractional part.

* * * *

Okay, now this leads to an important question – given two integers *a* and *b*, what's the largest possible difference between their quotient with and without this truncation rule?

Well, I won't show how here, but it's not too hard to puzzle out that the largest difference between *((double)a / b)* and *(a / b)* is *(b-1)/b* if *a* is positive, *-(b-1)/b* if *a* is negative. So a bit less than +/-1 at most.

* * * *

Here's another way to think of the meaning of 7 / 3 – 1 in both non-truncated and truncated contexts.

You can think of 7 / 3 – 1 as the 1 dimensional volume on the number line from 1 all the way up to 7 / 3. It's approximated in C# like this, with 7.0/3 for *n* and with *d* some utterly tiny value like, say, 10^-300:

```
static double Division( double n, double d ){
        double t = 0;
        for (double x = 1 + d; x <= n; x += d){
                t += d;
        }
        return t;
}
```

As *d* gets closer to 0, we get closer to the actual answer... but if *d* equals 0, we have an infinite loop and everything falls to pieces.

Our integer division, on the other hand, doesn't measure, it counts. Here, it counts every whole integer point on the number line between (but not including) 1 and 7/3. And how might we count those points? Simple – use **Division** with a *d* parameter of 1.

One way to think of this is that **Division** samples *n* with *d* sized samples, and as d gets smaller we get closer and closer to actually measuring *n*.

And in fact, our remainder is approximately Division(n, 10^-300) - Division(n, 1).

* * * *

So, to recap: when we divide an integer by an integer and require an integer answer, we introduce some error. That error is the remainder. We can put limits on how big that difference can possibly be.

Reasoning about this kind of error is a pretty common aspect of digitizing or quantizing or making information discrete. In fact, if you've worked as a performance-conscious programmer for any length of time, keeping these sorts of

issues in the back of your mind has probably become second nature.

* * * *

If you've stuck with me this far, now I'm going to show you something astonishing.
Let's take **Division** and make it recursive with a bit of extra logic. Here's our new function:

```
static double RecursiveDivision( double n, double d, int k ){
        double t = 0;
        for (double x = 1 + d; x <= n; x += d){
                t += d / k – d * RecursiveDivision( n / x, d, k+1 );
        }
        return t;
}
```

So this is nearly identical to **Division** from before, but with a parameter k added to the function signature, and the logic inside the loop a bit more complicated and recursive. Like before, the smaller *d* gets, the closer we get to our actual answer. Don't worry too much about what this function means; just make sure you have a sense of what it looks like it's doing. It doesn't look like it is too complicated.

So, here's an innocent enough sounding question: what's the largest possible difference between RecursiveDivision(n, 10^-300, 1) and RecursiveDivision(n, 1, 1)? Sounds similar to our remainder problem from before, doesn't it. Can you figure it out?

* * * *

Did you come up with an answer? Any guesses? How confident do you feel?
Really try chewing on this. It's a tricky problem. What kind of approaches would you take to put hard limits on the remainder value here?

* * * *

Okay. One last, leading question. Do you think that the difference between RecursiveDivision(n, 10^-300, 1) and RecursiveDivision(n, 1, 1) will always be less than $\frac{1}{8\pi}\sqrt{n}\log n$ for any *n* greater than or equal to 59? Can you figure out a way to prove it that would satisfy rigorous mathematicians?

* * * *

Well, if you can, you're about to have a really exciting set of months and years ahead of you. You'll immediately win a million dollars and your name will be remembered by humanity into the far, far future...
… and this is one of the very few instances where that's not hyperbole. It's actually true. Proving that the difference between RecursiveDivision(n, 10^-300, 1) and RecursiveDivision(n, 1, 1) is always less than $\frac{1}{8\pi}\sqrt{n}\log n$ for any *n* >= 59 is equivalent to proving the Riemann Hypothesis, possibly the most important open question in math.

```
for (double x = a; x <= n; x += d) t += f(x) * d;
```

$$\int_{a}^{n} f(x)\,dx$$

```
for (int x = a; j <= n; x++) t += f(x);
```

$$\sum_{x=a}^{\lfloor n \rfloor} f(x)$$

$$\text{Division, } \lim_{d \to 0} = \int_{1}^{n} dx$$

$$\text{Division}, d = 1 = \sum_{j=2}^{\lfloor n \rfloor} 1$$

$$\text{RecursiveDivision, } \lim_{d \to 0} = \int_{1}^{n} dx - \frac{1}{2} \int_{1}^{n} \int_{1}^{\frac{n}{x}} dy\,dx + \frac{1}{3} \int_{1}^{n} \int_{1}^{\frac{n}{x}} \int_{1}^{\frac{n}{xy}} dz\,dy\,dx - \frac{1}{4} \int_{1}^{n} \int_{1}^{\frac{n}{x}} \int_{1}^{\frac{n}{xy}} \int_{1}^{\frac{n}{xyz}} dw\,dz\,dy\,dx + \ldots$$

$$\text{RecursiveDivision, d = 1} = \sum_{j=2}^{\lfloor n \rfloor} 1 - \frac{1}{2} \sum_{x=2}^{\lfloor n \rfloor} \sum_{y=2}^{\lfloor \frac{n}{x} \rfloor} 1 + \frac{1}{3} \sum_{x=2}^{\lfloor n \rfloor} \sum_{y=2}^{\lfloor \frac{n}{x} \rfloor} \sum_{z=2}^{\lfloor \frac{n}{x \cdot y} \rfloor} 1 - \frac{1}{4} \sum_{x=2}^{\lfloor n \rfloor} \sum_{y=2}^{\lfloor \frac{n}{x} \rfloor} \sum_{z=2}^{\lfloor \frac{n}{x \cdot y} \rfloor} \sum_{w=2}^{\lfloor \frac{n}{x \cdot y \cdot z} \rfloor} 1 + \ldots$$

$$li(n) - \log\log n - \gamma$$
$$\Pi(n)$$