COP 4516 Spring 2025 Ind Contest #5: Mathematics (Solution Sketches)

Fact

As covered in class, the number of times a prime p divides into n! can be formally represented as

$$\left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \cdots$$

In code, this is just a while loop where we divide n by p, add the result to an accumulator and repeat until n goes down to 0. This runs pretty quickly (O(lg n) time). So, we can run a prime sieve to generate all primes upto n, and then for each of the primes, compute the sum shown above.

Perfection in Numbers

The first issue many people had with this problem is that they didn't read the bounds. The input value can be as large as 10^{12}. Thus, the input value can't fit in an int and trying to read it in as an int in Java (nextInt()) causes a run time error.

Once you realize you have to read in the value as a long, the next hurdle is that the standard brute force solution of checking each divisor up to half of n takes too long, since we can't do 5 x 10^{11} simple operations. But the key is realizing that divisors of numbers come in pairs, except for the square root in the case of perfect squares. Thus, we need to only check for divisors up to the square root of the input value and this loop runs at most 10^6 times. When we find a divisor, there are three cases: (a) The divisor is 1, just add 1 to the running sum since it's corresponding divisor n should't be counted, (b) The divisor, k is less than the square root of n, in which case we want to add both k and n/k to our running sum, or (c) The divisor k is such that k x k = n. In this case we only want to add k once, not twice.

Profits

This is exactly the MCSS problem taught in class. In this problem however, they specify that the contiguous subsequence must be non-empty. Thus, instead of setting the initial result to 0, it must be set to the first value in the list, just in case all values in the list are negative. Also, since there are up to 250,000 values in the list, the O(n) algorithm must be used.

Relatively Prime

The question is asking for the computation of the Euler Phi Function. The formula for this was given in class. We can either find the prime factorization of n or a list of each unique prime divisors of n. If we use the prime sieve to precompute primes less than 1,000,000 (there are about 78,000) of these, then we can obtain the necessary information for an input case in about 78,000 simple steps. As we are discovering primes, update our answer. There are two ways to do this. Initialize phi = n, and then each time you find a unique prime divisor of it, update phi by dividing it by p and multiplying it by (p – 1), which is the formula for the phi function that only utilizes the unique prime divisor list. Alternatively start with phi = 1, and then whenever a term p^k is encountered in the prime factorization, multiply phi by (p^k – p^{k-1}). If you do this route, don’t do any double computations, just everything in integers.