Problem Set 7 - Math

<u>Question D</u>

Two divisors was one that clicked relatively nicely, just took some time. From the bounds of 5 * 10^5 items in the array, with each item being 10^7, if we were to run standard factorisation over each of the elements, our final time complexity would be 5 * 10^5 * sqrt(10^7) = 1.58e9, much too slow to pass. So the observation here is that we should probably use the sieve of eratosthenes to get all of the factors.

We also, for each of the test cases, would want to compute the answer for the pair relatively quickly (faster than sqrt n) so as to not dominate the sieve in time complexity.

So a biiiig hint that this was leading to was that there was some sort of mathematical property that allowed us to take the factors from the sieve, and - in constant time - split them in a way that got me to the answer. Finally, helped along by one of my maths friends - I came to the answer. If you just partition the prime factors in any way, since the prime factors on both sides of the partition are coprime, there cannot be a number that divides both greater than 1. If you then sum both sides together, and then suppose the gcd of d1 + d2 and our number is greater than 1, there must be a number that divides both d1 and d2. But this is a contradiction with our earlier observation.

The proof by contradiction is written (semi) formally in the comments of the code - I'm not a math major so excuse my terrible explanations.

Ultimately, this means we just run the sieve to find all factors up until 1e7, then, when processing a test case, we choose the first prime, multiply it by it's multiplicity to get d1, and then divide it out of our main number to get d2.

One issue I ran into was that running the Sieve up until 10^7 to find ALL factors was quite memory intensive, and also, ran me into a time limit exceeded fail on test 23. But the thing to notice here is that we don't need ALL of the factors, we can partition the prime factors in any way in order to devise a d1 and d2. And thus, instead of storing a list of vectors, or a list of maps, I could just store and compute the any prime factor of each number in a 1d array/vector.