1. Give a formula for the number of positive divisors of a number \( n \) based on its factorization into primes. That is, if
\[
 n = p_1^{m_1} p_2^{m_2} \cdots p_k^{m_k}
\]
then determine how many divisors \( n \) has.

**ANSWER:** The divisors of \( n \) are precisely the integers whose prime factorization is of the form
\[
 n = p_1^{m_1} p_2^{m_2} \cdots p_k^{m_k}
\]
for some non-negative integers \( m_i \leq n_i \). So there are \( n_1 + 1 \) possible values for \( m_1 \) (it could be any of \( 0, 1, 2, \ldots, m_1 - 1, m_1 \)), and independently we may choose any of \( n_2 + 1 \) values for \( m_2 \), etc. The total number of divisors is then the total number of choices for the sequence \((m_1, m_2, \ldots, m_k)\), which is then
\[
(n_1 + 1)(n_2 + 1)\ldots(n_k + 1)
\]

2. Show that if \( a \) and \( b \) are coprime integers and \( a \cdot b \) is a perfect cube, then \( a \) and \( b \) are perfect cubes too. The corresponding statement for squares is almost true, but there’s a little subtlety; can you find it?

**ANSWER:** Obviously if an integer \( n \) has prime factorization
\[
 n = p_1^{m_1} p_2^{m_2} \cdots p_k^{m_k}
\]
then the \( r \)th power of \( n \) has prime factorization
\[
 n^r = p_1^{rm_1} p_2^{rm_2} \cdots p_k^{rm_k}
\]
and thanks to the Fundamental Theorem of Arithmetic that is the only decomposition \( n^r \) can have. So \( r \)th powers can be recognized from their prime decomposition: all exponents must be multiples of \( r \). So for example if \( a \) and \( b \) are positive integers whose product \( ab \) is a cube, then all the exponents in the prime factorization of \( ab \) are multiples of \( 3 \). On the other hand, the prime factorization of \( ab \) may be obtained by simply pasting together the factorizations of \( a \) and \( b \) separately, *because* those two integers are coprime. Thus each exponent in the decompositions of \( a \) and \( b \) are themselves multiples of \( 3 \), meaning \( a \) and \( b \) are perfect cubes.

Replace “cube” with “square”, and “3” with “2”, and the previous paragraph again stands. But the original question didn’t specify that the integers are positive; if for example \( a = -4 \) and \( b = -9 \) then \( ab \) is a perfect square but \( a \) and \( b \) are not!

3. Show that for every integer \( n > 1 \), \( n^3 + 1 \) is composite. (Hint: you may find a few examples to be instructive. Try \( n = 1, 2, 4, 6, \) and \( 16 \).)
**ANSWER:** For every \( n \), \( n^3 + 1 = (n + 1)(n^2 - n + 1) \). Since for every number \( n > 1 \) both of those factors are greater than 1, this provides a nontrivial factorization of \( n^3 + 1 \).

4. **Twin primes** are primes \( p \) and \( q \) which differ by 2. For example 11 and 13 are twin primes. Prove that there are infinitely many primes which are NOT part of a twin-prime pair. How many primes \( p \) are there for which \( p, p + 2, \) and \( p + 4 \) are all prime?

**ANSWER:** I postponed this problem until the following week so that you would have Dirichlet’s Theorem at your disposal. A prime \( p \) is part of a twin-prime pair if either \( p - 2 \) or \( p + 2 \) is also prime, so what we want in this problem is an infinite set of primes \( p \) for which both \( p - 2 \) and \( p + 2 \) are composite. We might, for example, look for primes \( p \) for which \( p - 2 \) is a multiple of 3 (other than 3 itself) and \( p + 2 \) is a multiple of 5 (other than 5 itself). In other words we want primes \( p \) (other than 5 or 3) for which

\[
p \equiv +2 \pmod{3} \quad \text{and} \quad p \equiv -2 \pmod{5}
\]

By the Chinese Remainder Theorem these two congruences together simply state that \( p \equiv 8 \pmod{15} \). Since \( \gcd(8, 15) = 1 \), Dirichlet’s Theorem guarantees there are infinitely many such primes. (The first few are 23, 53, 83, 113, 173, 223, . . .).

5. For each integer \( n \) let \( C_n \) denote the central binomial coefficient \( C_n = \binom{2n+1}{2n} \).

Compute \( C_0, C_1, C_2 \). Show that for every integer \( M \), \( \gcd(M, C_n) \) is divisible by all the prime divisors of \( M \) that lie between \( 2^n \) and \( 2^{n+1} \).

**ANSWER:** The definition of the binomial coefficient in terms of factorials may be written this way:

\[
m! (n - m)! \binom{n}{m} = n!
\]

If \( p \) is any prime less than or equal to \( n \) then it divides the number on the right, and hence must divide one of the three factors on the left. If on the other hand \( p \) is larger than both \( m \) and \( n - m \) then \( p \) will not divide \( m! \) nor \( (n - m)! \); in that case it must divide the binomial coefficient. In the special case that \( n = 2m \), this means the binomial coefficient is divisible by every prime which lies (strictly) between \( m \) and \( n \). In particular, my \( C_n \) above is divisible by every prime between \( 2^n \) and \( 2^{n+1} \). (That’s all the the primes which are \( n + 1 \) bits long when expressed in binary. That’s a lot of primes!)

Of course that means every prime divisor of \( M \) which is in that same range will then divide \( \gcd(M, C_n) \), as was to be shown.

The first few of these numbers are 1, 2, 6, 70, 12870, 601080390.

A small variation of the numbers \( C_n \) may be used instead; look up the Catalan numbers.

Let me comment about that parenthetical part. It’s very easy to find all the prime divisors of a number \( M \). Compute, in turn, each of the \( \gcd(M, C_1) \), \( \gcd(M, C_2) \), \( \gcd(M, C_3) \), . . .

These will report to you in turn (the product of) all of \( M \)’s 2-bit prime divisors, then its
3-bit prime divisors, then its 4-bit prime divisors, etc. And remember, it’s very easy to compute a \( \gcd \) using the Euclidean algorithm. (Roughly speaking there are only about \( \log_2(M) \) steps to each such \( \gcd \) computation.) And even though the numbers \( C_n \) get large fast, we don’t ever really need them: if \( C_n \equiv X \mod M \) then \( \gcd(M, C_n) = \gcd(M, X) \), so we never really need to work with numbers bigger than \( M \) itself. So I could write a very fast computer program to find the prime divisors of any integer \( M \) is I could just figure out a way to insert \( C_n \) quickly into the program in the first place, or more precisely to have my computer compute \( C_n \) (modulo \( M \)) in a relatively few steps. I’m thinking of numbers \( M \) of say a couple hundred digits; that means I might need \( C_{1000} \). What’s the fastest way to compute this number (modulo any integer \( M \), say)?