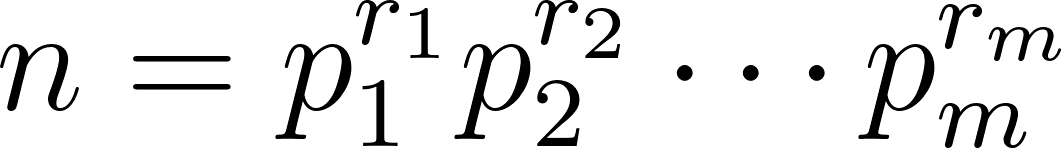
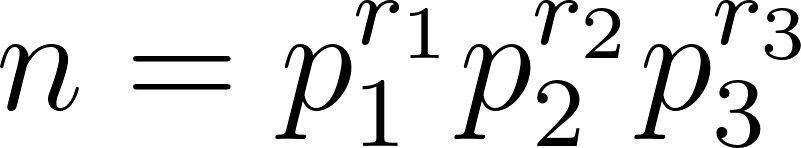
**Homework 1**

**Instructions:** Do as many of the problems as you like, but make sure to complete at least two (counting 1 as “three problems”, one for each part). Then the whole class will create a solution jointly (write below, or create a separate file; either is OK). One problem can have multiple solutions, so if your solution is different than the one already posted and you’d like to share yours with others, feel free to add yours. Make sure to separate different solutions to minimize confusion.

1. **a.** Given an integer [](https://www.codecogs.com/eqnedit.php?latex=n%3Dp_1%5E%7Br_1%7Dp_2%5E%7Br_2%7D%5Ccdots%20p_m%5E%7Br_m%7D#0), how many factors does *n* have? (You can consider a special case of [](https://www.codecogs.com/eqnedit.php?latex=n%3Dp_1%5E%7Br_1%7Dp_2%5E%7Br_2%7Dp_3%5E%7Br_3%7D#0) if you’d prefer to work with a slightly more concrete case)

* The number of factors of *n*, using the multiplication principle, is . This counts the number of integers whose prime factors are a subset of the prime factors of *n* and where the powers on those prime factors are less than or equal to the corresponding powers on the prime factors of *n*, while remaining nonnegative. Each prime factorization in this list will be unique so there will be no repeats of the integers that factorization corresponds to.

**b.** Suppose we are looking to find out which *n* have an odd number of factors? What does part **a** tell about *n*?

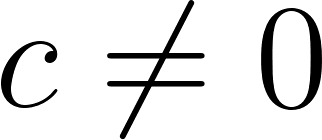
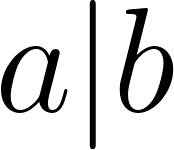
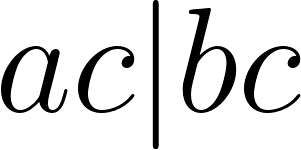
* Part **a** tells us that for *n* to have an odd number of factors, the product of one more than the exponents of the prime factorization of *n* must be odd. Thus all of the exponents of the prime factorization of *n* must be even, making *n* a perfect square.

**c.** Now let’s use the other approach of finding factors of a number: pairing a factor *r* with *n/r*. What does this tell us about *n* if *n* has an odd number of factors? Which method was easier in determining this property?

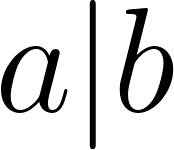
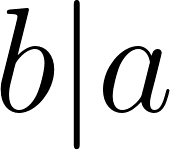
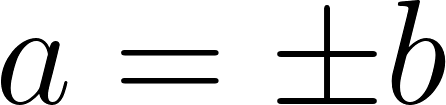
* If *n* has an odd number of factors, then some factor must have been paired with itself. Thus there exists some integer *r* such that and thus that . Thus if *n* has an odd number of factors then *n* is a perfect square. This method was easier in determining that fact.

1. If we know that *d* | *a* and *d* | *b*, what other divisibility relations can be concluded? List a few examples and prove the most general conclusion you can make.

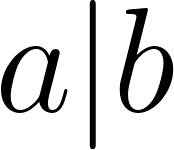
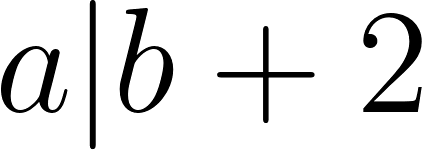
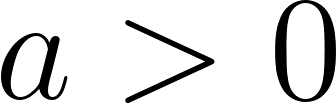
* for some integer k
* for some integer m
* for all integers *p, q*. the integers are closed under multiplication and addition so .
  + Thus, *d* divides all integer linear combinations of *a* and *b*.

1. Prove: If [](https://www.codecogs.com/eqnedit.php?latex=c%5Cneq%200#0), then [](https://www.codecogs.com/eqnedit.php?latex=a%7Cb#0) iff [](https://www.codecogs.com/eqnedit.php?latex=ac%7Cbc#0). (Iff is short for if and only if.)

* if and then
  + for some integer *k*
  + Multiply both sides by *c* to obtain ,
* if and then
  + for some integer k
  + (multiplicative cancellation or divide by *c*)

1. Prove: If [](https://www.codecogs.com/eqnedit.php?latex=a%7Cb#0) and [](https://www.codecogs.com/eqnedit.php?latex=b%7Ca#0), then [](https://www.codecogs.com/eqnedit.php?latex=a%3D%5Cpm%20b#0).

* for some integer *p*
* for some integer *q*
* Thus so
* Either and or and
* If then
* If then

1. Prove: If [](https://www.codecogs.com/eqnedit.php?latex=a%7Cb#0) and [](https://www.codecogs.com/eqnedit.php?latex=a%7Cb%2B2#0) and [](https://www.codecogs.com/eqnedit.php?latex=a%3E0#0), then [](https://www.codecogs.com/eqnedit.php?latex=a%3D1#0) or [](https://www.codecogs.com/eqnedit.php?latex=a%3D2#0).

* for some integer *k*
* for some integer *j*
* , ,
* So, since 2 is prime, either and or and
* Thus, or

1. Write a code to check if a|b is true or not. (Once you have a draft, feel free to Google to see if there’s a shorter solution.)

* (Python)
* def divides(a, b):
  + if a==0:
    - return False
  + if b<0 or a<0:
    - return divides(abs(a), abs(b))
  + while b > 0
    - b -= a
  + return b==0
* shorter but same idea (felt a bit like cheating to use the mod operator):
* def divides(a, b):
  + if a==0:
    - return False
  + return b % a == 0

def divides(a, b):

if a==0:

return False

if b<0 or a<0:

return divides(abs(a), abs(b))

while b > 0:

b -= a

return b==0