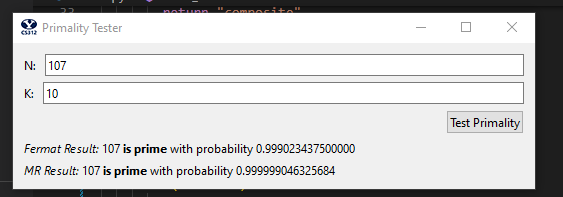
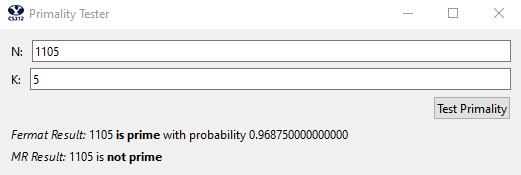
1. 
2. asdf
3. What I did to test where fermant’s and miller-rabin’s algorithms gave different answers was I started to test the Carmichael numbers. I noticed that as k got bigger (10 or so) then both of the algorithms caught them as not prime. But when the k was lower, like 5, then usually the format’s algorithm got fooled but the miller-rabin algorithm caught the number correctly. This is the case because fermant fails when the “a” value chosen is relatively prime to the N value. Choosing just a few “a” values increases the likely hood of it being relatively prime, but as soon as it is not relatively prime, then fermant catches it. So the bigger the k, the more likely one of these will not be relatively prime.
4. Fermant time and space complexity
   1. The first step is figuring out the mod\_exp complexity = n^3
   2. Miller-robin -> Fermant \* n = n^4
5. The Equation I got for the probability of fermant was 1-.5^k. This equation came from the fact that there is a .5 chance that any chosen “a” value will be prime. Then the next time we pick an “a” value, it will be another .5 chance of our .5 chance. This perpeturates k times or .5^k. But this is the probability that the fermant algorithm gave us the wrong answer. To get the probability that it gave us the right answer, we just do the inverse (1-.5^k). For the Miller-Rabin algorithm, we know that there is a .75 chance that for composites it gives us correct results. This means that there is a .25 chance that a composite number was incorrectly identified. Each subsequent “a” value chosen is .25 chance, or .25^k. Inverting this to get the probability that we did it right is 1-.25^k.

import random

def prime\_test(N, k):

  # This is main function, that is connected to the Test button. You don't need to touch it.

  return fermat(N,k), miller\_rabin(N,k)

def mod\_exp(x, y, N):

  if y == 0:

    return 1

  z = mod\_exp(x, y // 2, N)

  even = y % 2

  if even == 0:

    return z \* z % N

  else:

    return z \* z \* x % N

def fprobability(k):

    return 1 - .5\*\*k

def mprobability(k):

    return 1 - .25\*\*k

def fermat(N,k):

    for i in range(k):

      a = random.randint(2, N - 1)

      mod = mod\_exp(a, N-1, N)

      if mod != 1:

        return "composite"

    return "prime"

def miller\_rabin(N,k):

  for i in range(k):

    a = random.randint(2, N - 1)

    e = N-1

    while True:

      mod = mod\_exp(a, e, N)

      if (mod != 1):

        #mod == -1 and mod = N - 1 are the same

        if mod != -1 and mod != N - 1:

          return "composite"

        else:

          break

      #Loop until we cannot divide by two anymore

      if e % 2 != 0:

        break

      e = e // 2

  return "prime"