COMPUTER EXPERIMENTATION #4 – MATH 5405
SPRING 2016

DUE: THURSDAY, FEBRUARY 25TH

We will implement Miller-Rabin's primality test and some related methods.

I want to emphasize that you should be creating functions (using the `def` command). You can then reuse your code much more easily.

We start with Miller-Rabin.

(1) First create a function which figures out how many powers of 2 divide a number. I made mine something like this.

```
def power2Count(n):
     while (n%2 == 0):
        n = n/2
        i = i+1
    return [i - 1, n]
```

See if you can figure out why returned what I did.

(2) Now let's make a function `RabinMiller` which checks if a particular number a proves that n is composite (via Rabin-Miller). It should return True if the number is composite, and return False the test is inconclusive. I started my function like this.

```
def RMCompositeTest(a,n):
    shortList = power2Count(n-1)
    k = shortList[0]
    q = shortList[1]
    #note then that n-1 = 2^k * q
    ...
```

Then I checked if $a^q \equiv_n 1$ (if so, I return False). Next I did a loop and checked whether $a^{2^i q} \equiv_n -1$. If any of those occurred, then I returned False as well. At the end of my function, I returned True.

(3) Now we need to make our Rabin-Miller test effective. We need a way to generate random numbers (random a values) to plug into our RMCompositeTest function. The `random` package has a `randint` function that does exactly this. Try somethings like the following.

```
>>> import(random)
>>> random.randint(2,5)
>>> random.randint(2, 928751)
>>> random.randint(2, 576980719386713897619034671349067134890 71)
>>> random.randint(2, 576980719386713897619034671349067134890 71)
>>> random.randint(2, 576980719386713897619034671349067134890 71)
```

(4) Now let's write a function that will generate a number of random a values and check them all to see if n is proved composite. I started mine like this.

```
import random

def runRMKTimes(K, n):
    for j in range(0,K):
```

```
a = random.randint(2,n-1)
...
```

Once you get this working (ie, after you try some small primes), why don't you see how long your computer takes to run 1000 Rabin-Miller tests on the integer

$$2726762164912959739595080152067187581 13.$$

Or better yet on

$$6328447104016415844401817573993636478486114913058969763623400180026367594 1527.$$

Remember, if your random integers are really random, this should prove to within

$$(1 - 0.25^{1000}) \cdot 100\%$$

certainty that these numbers are prime.

(5) Now, let's make a function that finds the next number that is probably prime after a given number. I made mine so as to take a value, and then just check every odd number after it with C Rabin-Miller tests (user specified). Here's how I started mine.

```
#I pass it the number C and the number n.
# It tries to find the next prime after n by doing C Rabin-Miller tests.
def findNextPrime(C, n):
    if (n % 2 == 0):
        n = n+1
        #if n is even, make it odd, we will incremement by 1
        flag = True
        while (flag == True):
            ...
            n = n+2
        return n-2
```

Play around with it, can you find some big primes?