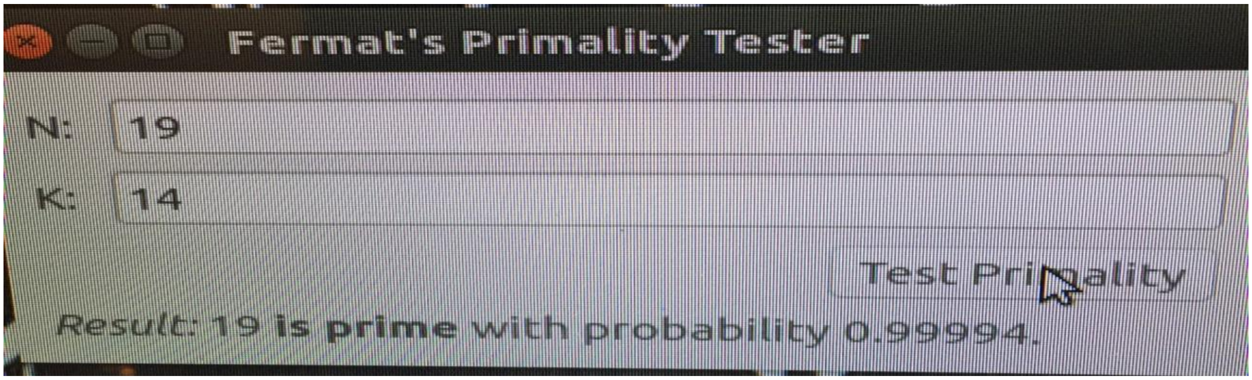
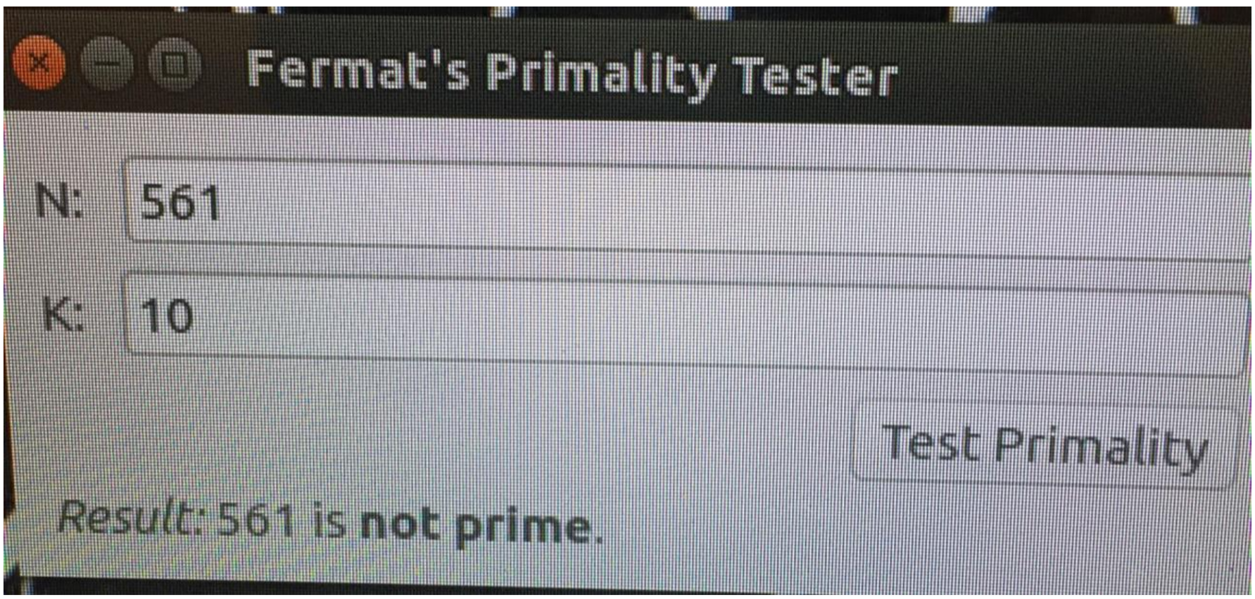
1. **Screenshots**

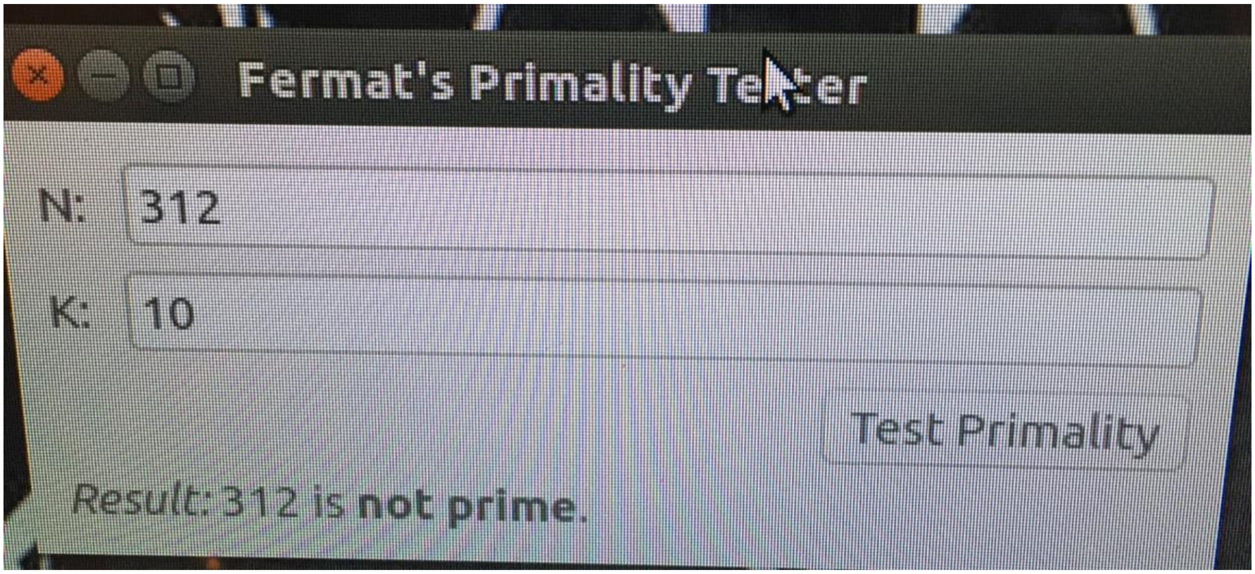
<u>Test of small prime number</u>



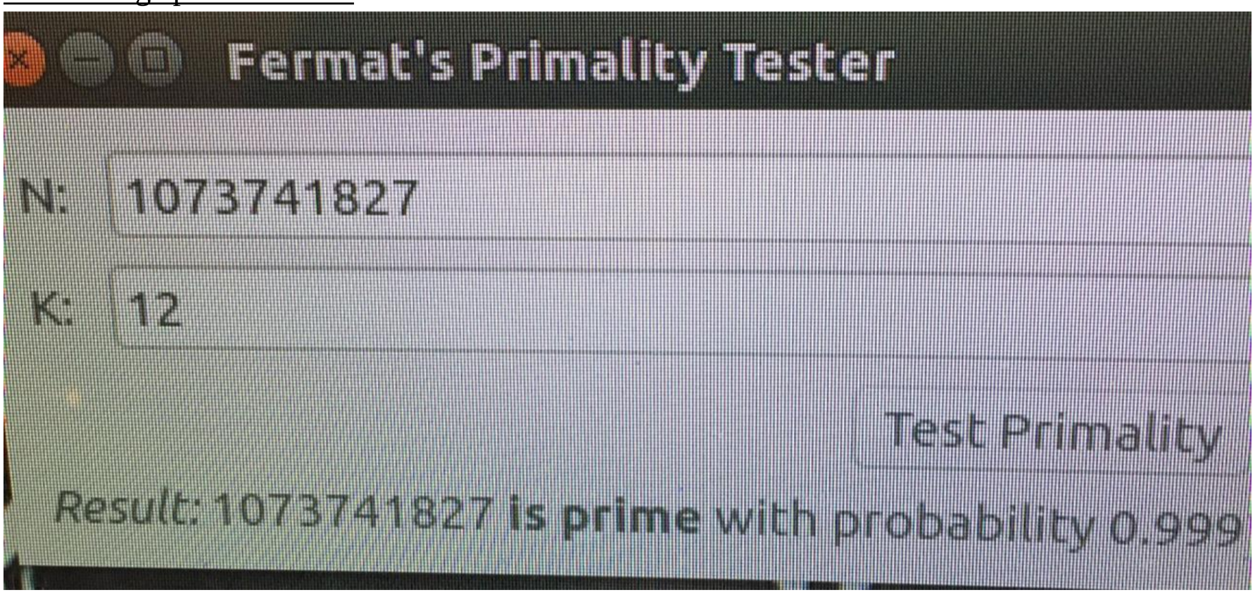
<u>Test of carmichael number</u>



<u>Test of composite number</u>



Test of large prime number



2. **All code**

```
import random
```

```

#Assuming each has n-bits

#O(tn) + O(nlog(N)) + O(n)
def carmichael_test(N):
	b = 2 #space: O(n) bits
	while b < N: # will happen t = N-b-1 times: O(tn)
		#check if b is relatively prime to N and passes mod_exp function, if so it is prime

		#gcd(euclids) complexity: O(log(n*b + n* N))
		#		space: O(2n) = O(n)
		#mod_exp: complexity: O(N*n*log(N*n))
		#	space: O(3n) = O(n) but the recursive call itself is O(n^2)
		if(gcd(b,N) == 1 and mod_exp(b, N-1, N)):
			return 0;
		b = b+1			#space: O(n)

	return 1;

#complexity: O(xn) * O(nlog(N*n))
#space: O(2n) = O(n)
def prime_test(N, k):

	#space: O(n*k)
	rand_numbers = random.sample(range(2, N-2), k)

#	for x in rand_numbers:This was used to verify the numbers were not repeated
#		print(x)

	for x in rand_numbers:			#complexity: O(x * n)
		if(mod_exp(x, N-1, N) != 1):	#complexity: O(nlog(N*n))
			return 'composite'	#space: O(n) but recursive call is O(n^2)

	if(carmichael_test(N)):			#complexity: O(tn) + O(nlog(N)) + O(n)
		return 'composite'
	return 'prime'

#complexity: O(log(n * N))
#space: O(3n) = O(n)
def mod_exp(x, y, N):
	if y == 0:
		return 1
	z = mod_exp(x, y//2, N)

	if y % 2 == 0:			#y is even
		return (z*z) % N		#complexity for multiplication: O(z*n)^2 = O(n^2)
	else:				#y is odd
		return x * (z*z) % N	#complexity: O(x*n +z*n)^2 = O(n)

#Dr. Farrell said not to do complexity for probability
def probability(k):
    return 1 -(1/(2**k))


#Perform Euclids algorithm to get gcd
#Complexity: O(log(a+b)) for  Euclids algorithm
#space: O(a*n + b*n) = O(n)
def gcd(a,b):
	if(a < b):
		return gcd(b,a)			#same complexity & space
	if(a % b == 0):
```

```
        return b

    return gcd(b, a % b)
```

3. **Time and space complexity** (assuming n bits)

<u>Prime Test</u>
Time complexity: O(n) * O(nlog(N*n))
Space complexity: O(n*k) + O(n) (would be + O(n^2) if we count the recursive calls

<u>Carmichael Test</u>
Time complexity: O(tn) + O(nlog(N)) + O(n)
Space complexity: O(n^2) with recursive calls

<u>Mod_exp</u>
Time complexity: O(nlog(N*n))
Space complexity: O(3n) = O(n)

<u>Gcd Euclids Algorithm</u>
Time complexity: O(log(a*n+b*n)
Space complexity: O(a*n + b*n) = O(n)

4. **"Discuss the equation you used to compute the probability of correctness *p* that appears in the output."**

For any Prime number 'p' and any 'a' in Z_p\{0,1...p-1\} the given probability that using our modexp() function on a random 'a' which is in the set Z_p will equal 1 is ½. In order to increase our probability, we pick more values of 'a' from the set Z_p. The probability increases with each 'a' chosen because $(½)^n$ (where n is the number of 'a's' chosen from the set Z_p) results in a smaller number as n increases. The result is subtracted from 1 to give us a percentage value.
$1 - (½)^n$