**By: Pawan Acharya**
**1/25/2022**
**Project 1**

1. **Modular exponentiation:**
   The exponentiation performed over modulus is modular exponentiation. It is necessary to compute x y mod N for values of x, y, and N that are several hundred bits long. So, modular exponentiation technique helps us find the values of x, y, and N. The pseudocode for modular exponentiation is:

   function modexp(x, y, N):
   Input: Two n-bit integers x and N, an integer exponent y
   Output: x^y mod N
   y mod N
   if y = 0: return 1
   z = modexp(x, y/2, N)
   if y is even:
   return z^2 mod N
   else:
   return x · z^2 mod N
   (Reference: Book)

Now, my implementation of modular exponentiation in code(in python) is given by:

```
def mod_exp(x, y, N):
    if y == 0: return 1
    z = mod_exp(x, y // 2, N)          >>Recursively makes O(N) calls
    if y % 2 == 0:                     >>Each recursive calls make O(N^2)
        return (z * z) % N
    else:
        return x % N * (z * z) % N
```

Looking at the time complexity, this algorithm makes O(n) recursive calls, and each of them takes O(n^2) time, so the complexity is O(n^3). So, the overall complexity is O(N^3).

2. **Fermat's Test algorithm for Primality test:**

Fermat developed a way to test for primality using a probabilistic algorithm. This test is a primality test, giving a way to test if a number is a prime number, using Fermat's little theorem and modular exponentiation.
Fermat's Little Theorem states that if a is relatively prime to a prime number p, then $a^{N-1} \equiv 1 \bmod N$
The pseudocode for Fermat's algorithm is:
function primality(N):
Input: Positive integer N
Output: yes/no
Pick a positive integer a < N at random

if a^(N−1) ≡ 1 (mod N):
return yes
else:
return no

In a loop this algorithm is given as:
function primality2(N):
Input: Positive integer N
Output: yes/no
Pick positive integers a1, a2, . . . , ak < N at random
if $a_i$ ^(N−1)≡ 1 (mod N) for all i = 1, 2, . . . , k:
return yes
else:
return no

(Reference: Book)

My implementation of code in Python is as follows:

```
def fermat(N, k):
    for i in range(1, k):                    Note that>>root N congruent to log N
        a = random.randint(2, N - 1)
        if mod_exp(a, N - 1, N) != 1:        >>Making k recursive calls, requires log(N-1)
complexity at each level
            return 'composite'
    return 'prime'
```

As seen here, the probability of error drops exponentially fast, and can be driven arbitrarily low by choosing k large enough. So, if it is prime it returns prime with a probability of 1-½^k.
The complexity behind this algorithm: I think that since it loops for k times, the complexity must depend on k parameter and finding a^N-1 requires log (N-1) .Thus, This means that
For the overall algorithm the time complexity seems to be O(k log^N).

### 3. **Miller-Rabin primality test:**

Miller-Rabin is also a probabilistic primality test that helps to determine if a number is composite or probably prime with some probability.In this algorithm, the process gradually processes by repeatedly taking the square root of a number that is ≡ 1 (mod N).

My implementation of this algorithm in python is :

```
def miller_rabin(N, k):
    c=0
    d = N - 1
    while d % 2 == 0:
        c += 1
        d //= 2
    for i in range(k):   >>k times the first loop means it will go k loops for the worst case
```

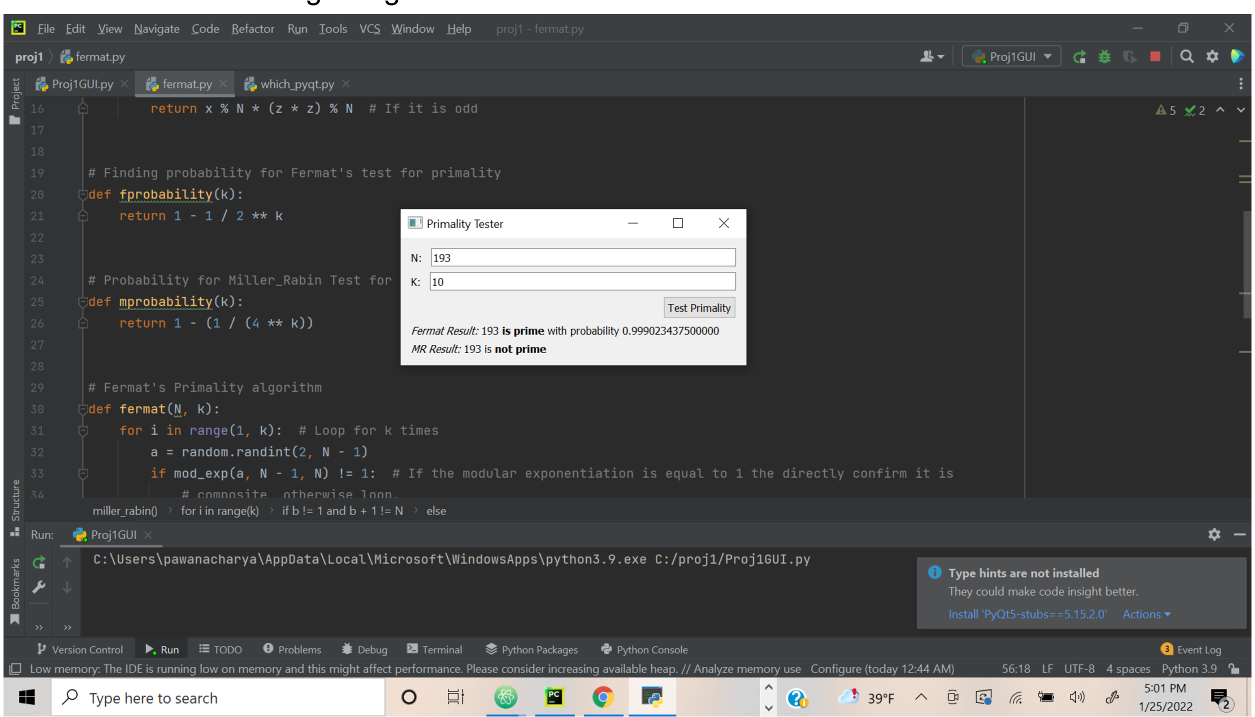
```
        a = random.randint(2, N-2)
        b = mod_exp(a, d, N)       >>>>Log (N-1) to calculate
        if b!=1 and b+1 !=N:
            for j in range(1,c):
                if mod_exp(b,2, N)!= 1:    >>>repeating squaring thus the complexity is O(k(logN)^3)
                    return 'composite'
                elif b!= N-1:
                        break
            else:
                return 'composite'
    return 'prime'
```

This algorithm uses most of the code of the Fermat's test.

I think that the Complexity of this algorithm is : O(k(log N)^3)

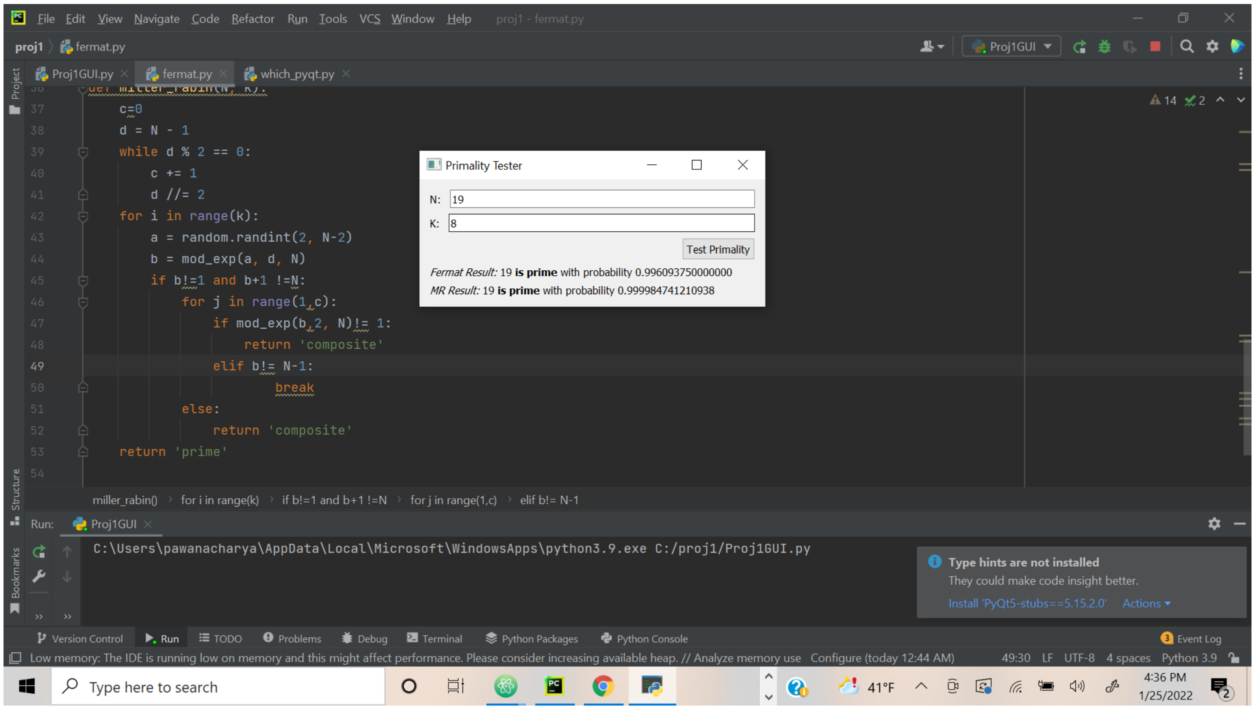
Agreement and Disagreement of the results on both algorithms:
For some inputs, the Rabin Miller test doesn't agree with Fermat's test. For eg: 193
But for most of the other inputs I entered, both agree with variety in probability.

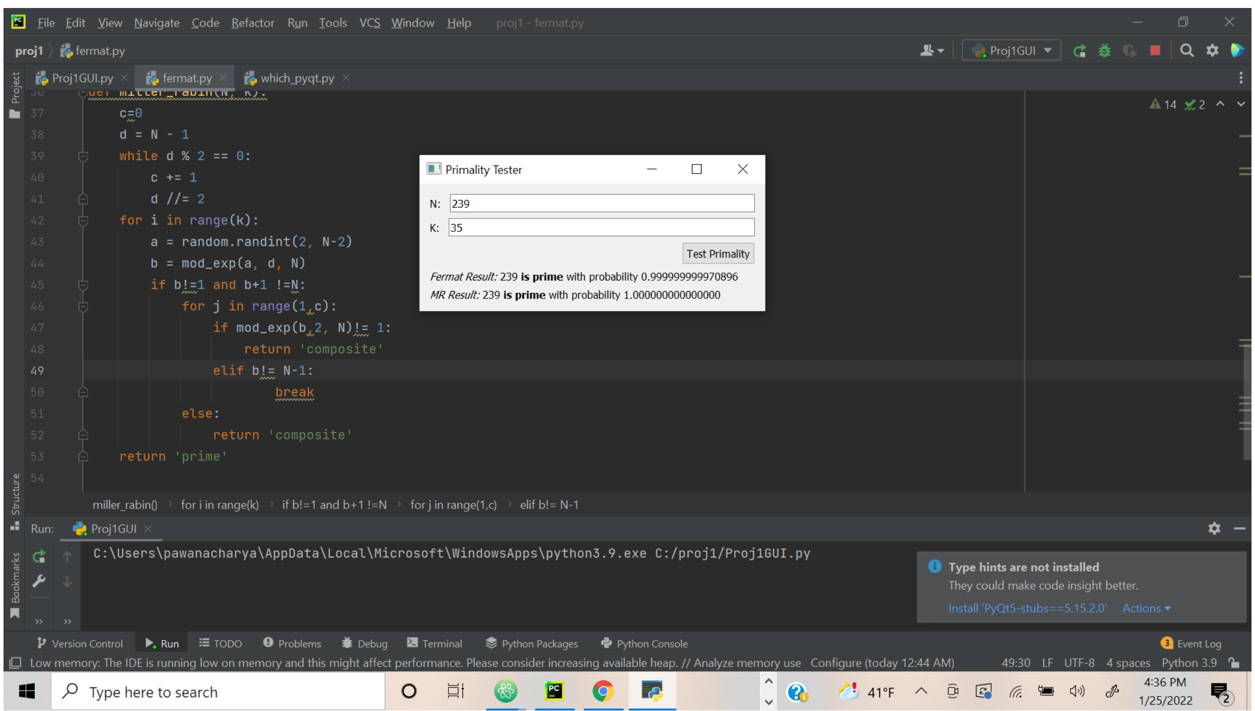
One screenshot showing disagreement is :

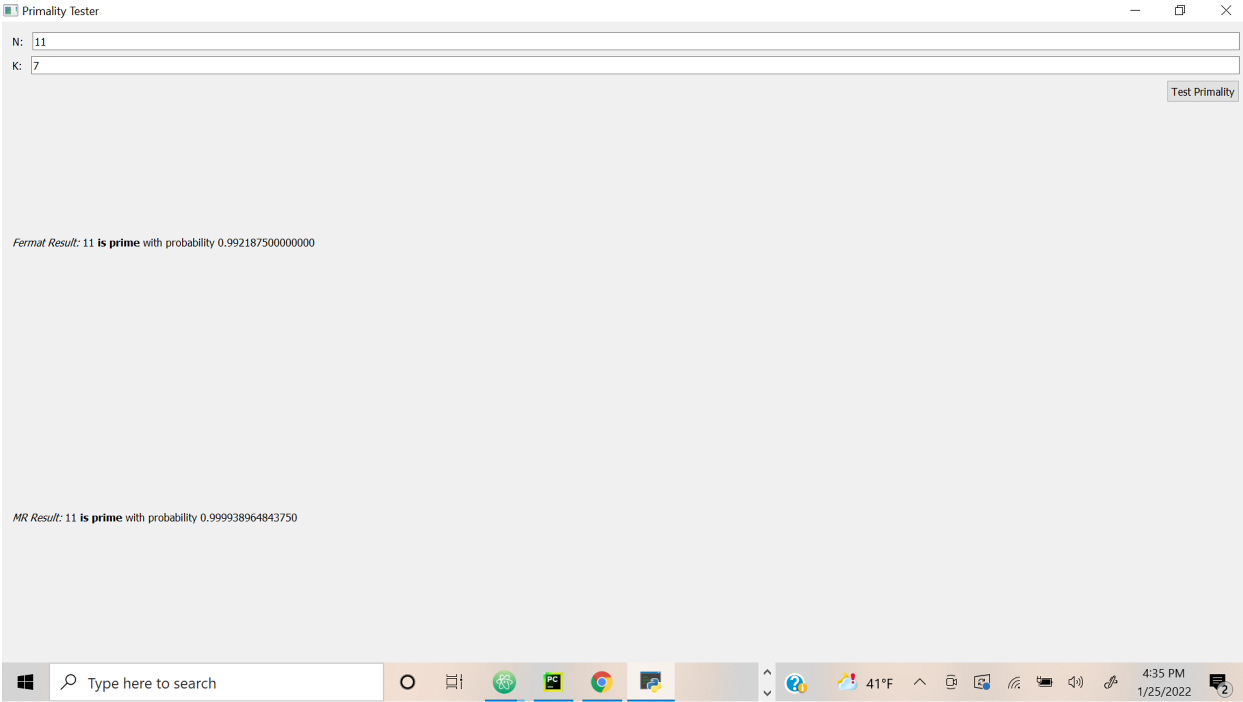


Discussion of probability in both cases:
For composites, for at least 3/4 of the possible choices for *a* , this will not be the case---either the initial test will not equal 1 (mod N).
The probability for being prime in Fermat's test is : $1-½^k$
Meaning that we can be sure that it can be prime with given probability.
Similarly, For Miller_Rabin's test: $1-¼^k$.

Some screenshot of my tests are :


File Edit View Navigate Code Refactor Run Tools VCS Window Help proj1 - fermat.py
proj1 fermat.py
Proj1GUI.py fermat.py which_pyqt.py
c=0
d = N - 1
while d % 2 == 0:
c += 1
d //= 2
for i in range(k):
a = random.randint(2, N-2)
b = mod_exp(a, d, N)
if b!=1 and b+1 !=N:
for j in range(1,c):
if mod_exp(b,2, N)!= 1:
return 'composite'
elif b!= N-1:
break
else:
return 'composite'
return 'prime'
Primality Tester
N: 19
K: 8
Test Primality
Fermat Result: 19 is prime with probability 0.996093750000000
MR Result: 19 is prime with probability 0.999984741210938
miller_rabin() for i in range(k) if b!=1 and b+1 !=N for j in range(1,c) elif b!= N-1
Run: Proj1GUI
C:\Users\pawanacharya\AppData\Local\Microsoft\WindowsApps\python3.9.exe C:/proj1/Proj1GUI.py
Type hints are not installed
They could make code insight better.
Install 'PyQt5-stubs==5.15.2.0' Actions
Low memory: The IDE is running low on memory and this might affect performance. Please consider increasing available heap. // Analyze memory use Configure (today 12:44 AM)
Type here to search
41°F
4:36 PM
1/25/2022


Primality Tester
N: 239
K: 35
Test Primality
Fermat Result: 239 is prime with probability 0.999999999970896
MR Result: 239 is prime with probability 1.000000000000000
Run: Proj1GUI
C:\Users\pawanacharya\AppData\Local\Microsoft\WindowsApps\python3.9.exe C:/proj1/Proj1GUI.py
4:36 PM
1/25/2022


Primality Tester
N: 11
K: 7
Test Primality
*Fermat Result:* 11 **is prime** with probability 0.992187500000000
*MR Result:* 11 **is prime** with probability 0.999938964843750
Type here to search
41°F
4:35 PM
1/25/2022