JANUARY 23, 2025

ASSIGNMENT 1

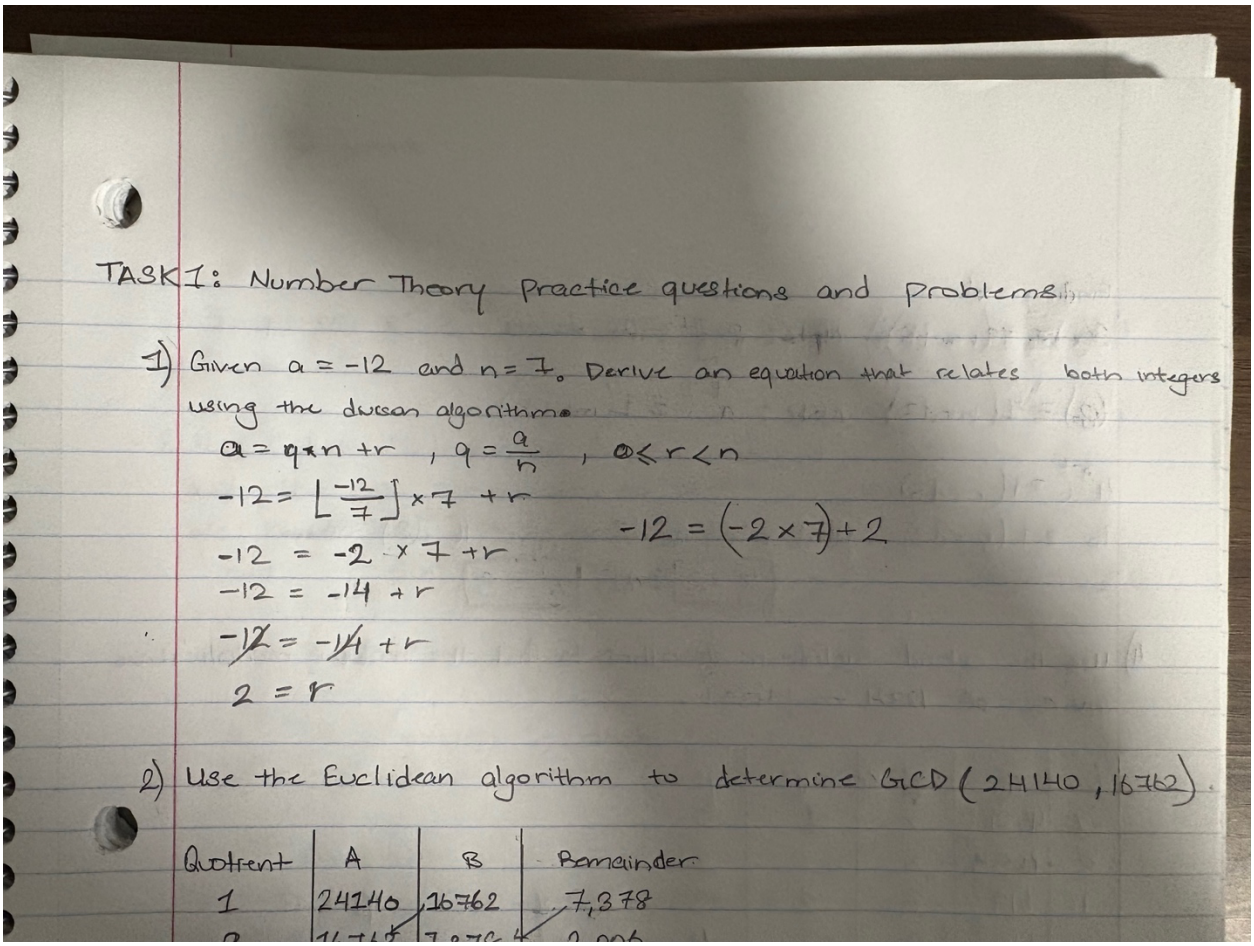
COMP – 7402 CRYPTOLOGY

BY: ARYAN JAND
TEACHER: ASHKAN JANGODAZ

Assignment 1 (COMP 7402)

Task 1: Number Theory Practice Question and Problems

Q1: Given a = -12 and n =7. Derive an equation that relates both integers using the division algorithm.



Q2: Use the Euclidean algorithm to determine GCD (24140, 16762).

$-12 = \left\lfloor \frac{-12}{7} \right\rfloor \times 7 + r$

$-12 = -2 \times 7 + r$

$-12 = -14 + r$

$-12 = -14 + r$

$2 = r$

$-12 = (-2 \times 7) + 2$

2) Use the Euclidean algorithm to determine GCD (24140, 16762).

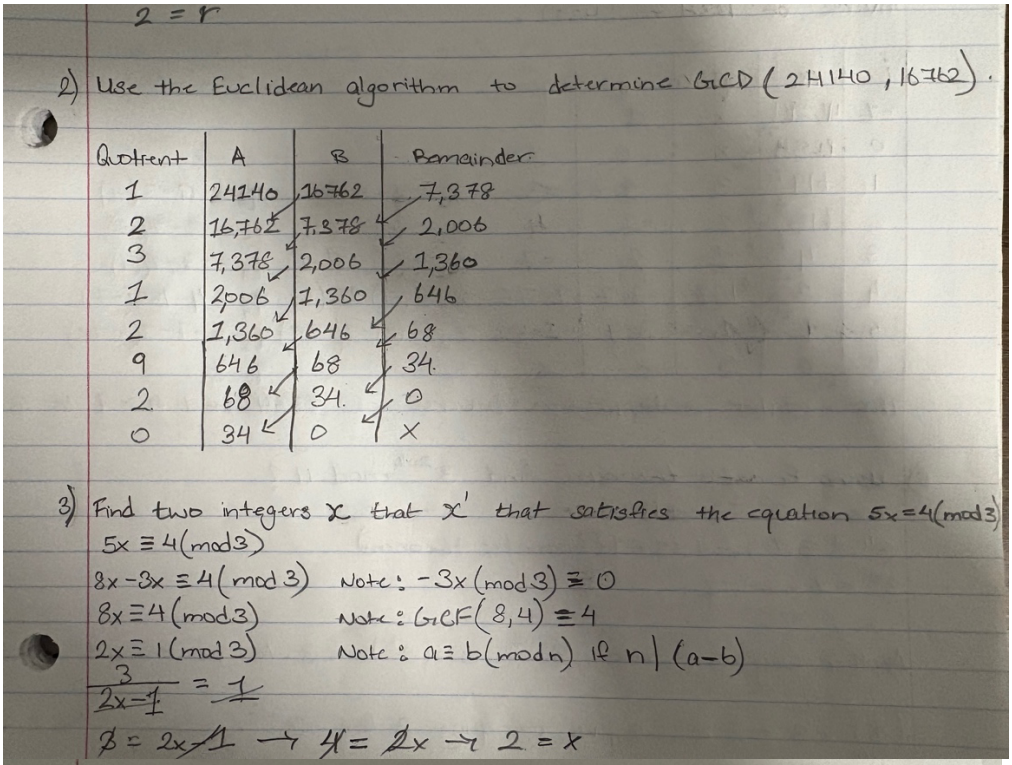
Quotient	A	B	Remainder
1	24140	16762	7,378
2	16,762	7,378	2,006
3	7,378	2,006	1,360
1	2,006	1,360	646
2	1,360	646	68
9	646	68	34
2	68	34	0
0	34	0	X

3) Find two integers x that x' that satisfies the equation $5x = 4 \pmod 3$

$5x \equiv 4 \pmod 3$

$8x - 3x \equiv 4 \pmod 3$ Note: $-3x \pmod 3 \equiv 0$

Q3: Find two integers x and x′ that satisfies the equation $5x \equiv 4 \pmod 3$.



2 = r

2) Use the Euclidean algorithm to determine GCD (24140, 16762).

Quotient	A	B	Remainder
1	24140	16762	7,378
2	16,762	7,378	2,006
3	7,378	2,006	1,360
1	2,006	1,360	646
2	1,360	646	68
9	646	68	34
2	68	34	0
0	34	0	x

3) Find two integers x that x′ that satisfies the equation $5x \equiv 4 \pmod 3$

$5x \equiv 4 \pmod 3$

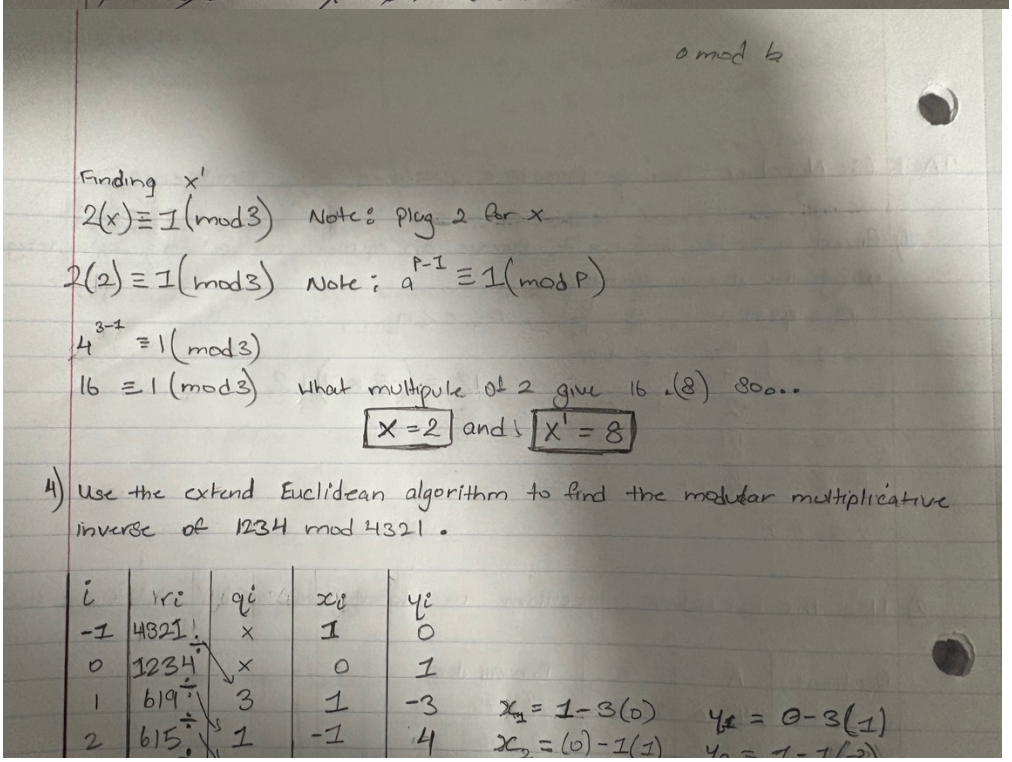
$8x - 3x \equiv 4 \pmod 3$ Note: $-3x \pmod 3 \equiv 0$

$8x \equiv 4 \pmod 3$ Note: GCF(8,4) ≡ 4

$2x \equiv 1 \pmod 3$ Note: $a \equiv b \pmod n$ if $n \mid (a-b)$

$\frac{3}{2x=1} = 1$

$3 = 2x - 1 \rightarrow 4 = 2x \rightarrow 2 = x$



a mod b

Finding x′

$2(x) \equiv 1 \pmod 3$ Note: Plug 2 for x

$2(2) \equiv 1 \pmod 3$ Note: $a^{P-1} \equiv 1 \pmod P$

$4^{3-1} \equiv 1 \pmod 3$

$16 \equiv 1 \pmod 3$ What multiple of 2 give 16 .(8) 8o..

x = 2 and x′ = 8

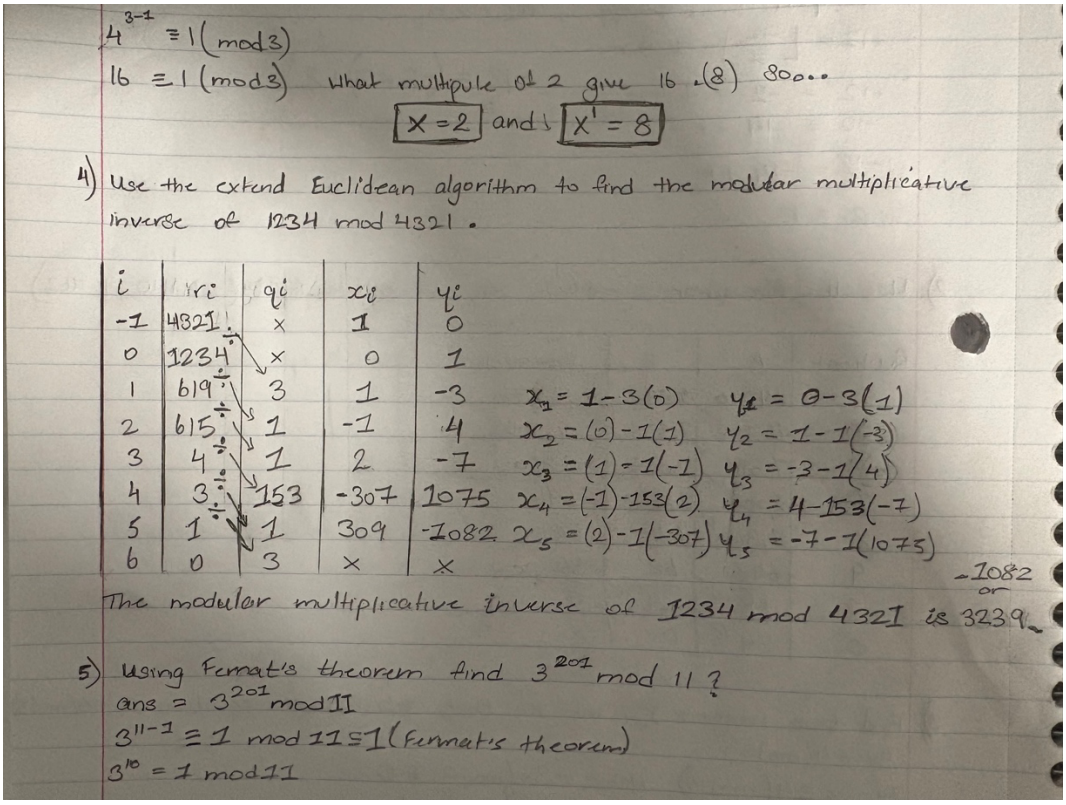
4) Use the extend Euclidean algorithm to find the modular multiplicative inverse of 1234 mod 4321.

i	ri	qi	xi	yi
-1	4321	x	1	0
0	1234	x	0	1
1	619	3	1	-3
2	615	1	-1	4

$x_1 = 1 - 3(0)$ $y_1 = 0 - 3(1)$

$x_2 = (0) - 1(1)$

Q4: Use the extend Euclidean algorithm to find the modular multiplicative inverse of 1234 mod 4321.



$4^{3-1} \equiv 1 \pmod 3$

$16 \equiv 1 \pmod 3$ What multiple of 2 give 16 (8) 800...

$x = 2$ and $x' = 8$

4) Use the extend Euclidean algorithm to find the modular multiplicative inverse of 1234 mod 4321.

i	ri	qi	xi	yi
-1	4321	x	1	0
0	1234	x	0	1
1	619	3	1	-3
2	615	1	-1	4
3	4	1	2	-7
4	3	153	-307	1075
5	1	1	309	-1082
6	0	3	x	x

$x_1 = 1 - 3(0)$ $y_1 = 0 - 3(1)$

$x_2 = (0) - 1(1)$ $y_2 = 1 - 1(-3)$

$x_3 = (1) - 1(-1)$ $y_3 = -3 - 1(4)$

$x_4 = (-1) - 153(2)$ $y_4 = 4 - 153(-7)$

$x_5 = (2) - 1(-307)$ $y_5 = -7 - 1(1075)$

-1082 or

The modular multiplicative inverse of 1234 mod 4321 is 3239.

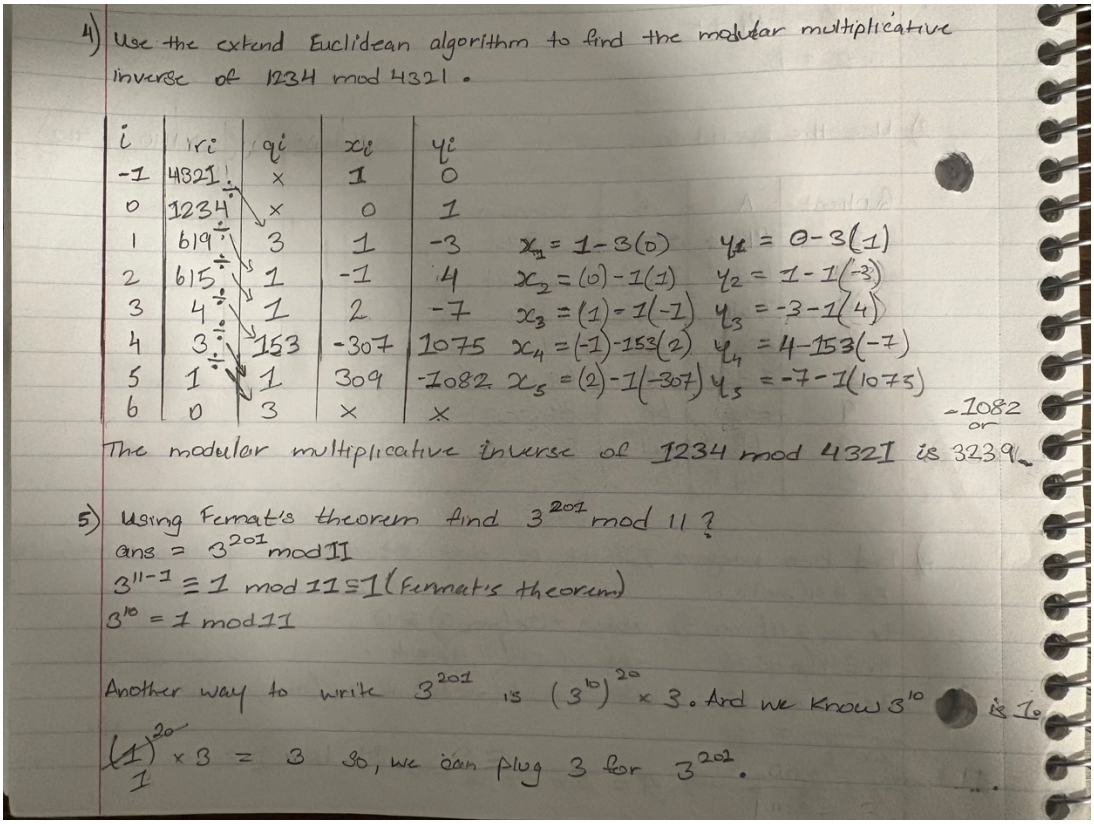
5) Using Fermat's theorem find $3^{201} \bmod 11$?

ans = $3^{201} \bmod 11$

$3^{11-1} \equiv 1 \bmod 11 \equiv 1$ (Fermat's theorem)

$3^{10} = 1 \bmod 11$

Q5: Using Fermat's theorem find 3201 mod 11.



4) Use the extend Euclidean algorithm to find the modular multiplicative inverse of 1234 mod 4321.

i	ri	qi	xi	yi
-1	4321	x	1	0
0	1234	x	0	1
1	619	3	1	-3
2	615	1	-1	4
3	4	1	2	-7
4	3	153	-307	1075
5	1	1	309	-1082
6	0	3	x	x

$x_1 = 1-3(0)$ $y_1 = 0-3(1)$
$x_2 = (0)-1(1)$ $y_2 = 1-1(-3)$
$x_3 = (1)-1(-1)$ $y_3 = -3-1(4)$
$x_4 = (-1)-153(2)$ $y_4 = 4-153(-7)$
$x_5 = (2)-1(-307)$ $y_5 = -7-1(1075)$

-1082 or 3239

The modular multiplicative inverse of 1234 mod 4321 is 3239.

5) Using Fermat's theorem find 3^{201} mod 11?

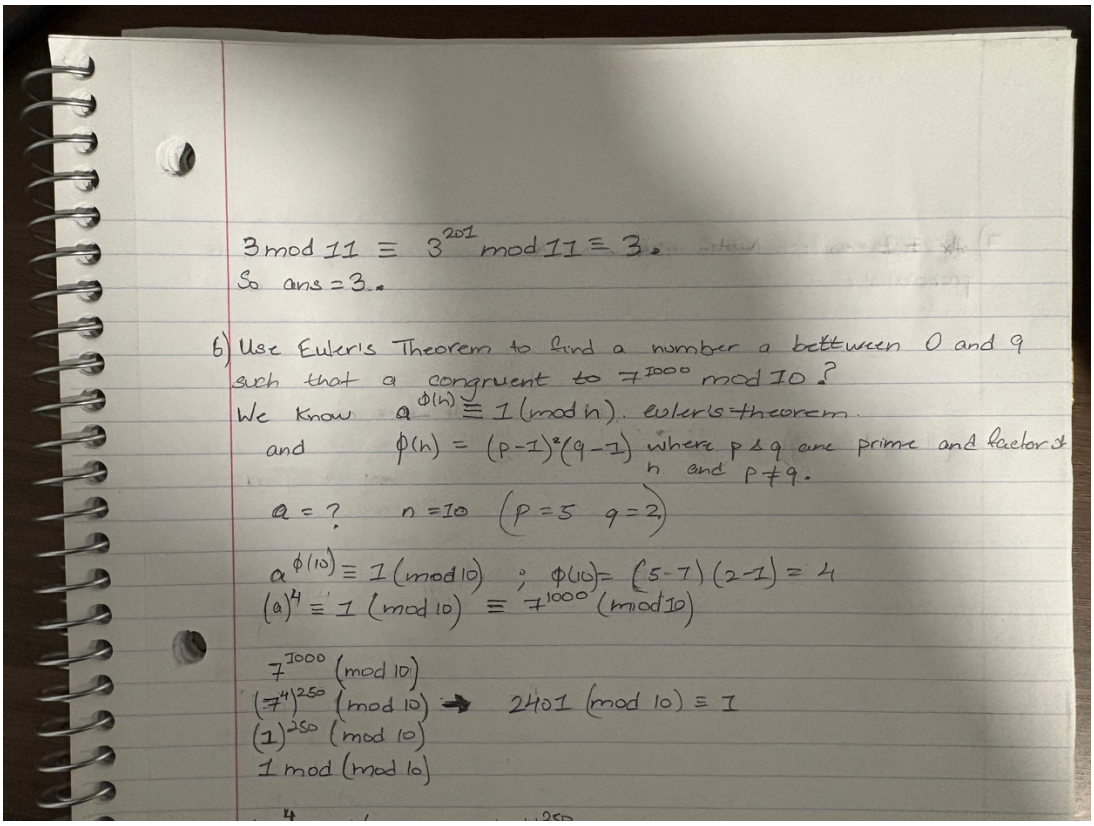
ans = 3^{201} mod 11

$3^{11-1} \equiv 1 \bmod 11 \equiv 1$ (Fermat's theorem)

$3^{10} = 1 \bmod 11$

Another way to write 3^{201} is $(3^{10})^{20} \times 3$. And we know 3^{10} is 1.

$(1)^{20} \times 3 = 3$ So, we can plug 3 for 3^{201}.



$3 \bmod 11 \equiv 3^{201} \bmod 11 \equiv 3$.

So ans = 3.

6) Use Euler's Theorem to find a number a between 0 and 9 such that a congruent to 7^{1000} mod 10?

We know $a^{\phi(n)} \equiv 1 \pmod n$. euler's theorem

and $\phi(n) = (p-1)^*(q-1)$ where p & q are prime and factors of n and $p \neq q$.

$a = ?$ $n = 10$ $(p = 5 \quad q = 2)$

$a^{\phi(10)} \equiv 1 \pmod{10}$; $\phi(10) = (5-1)(2-1) = 4$

$(a)^4 \equiv 1 \pmod{10} \equiv 7^{1000} \pmod{10}$

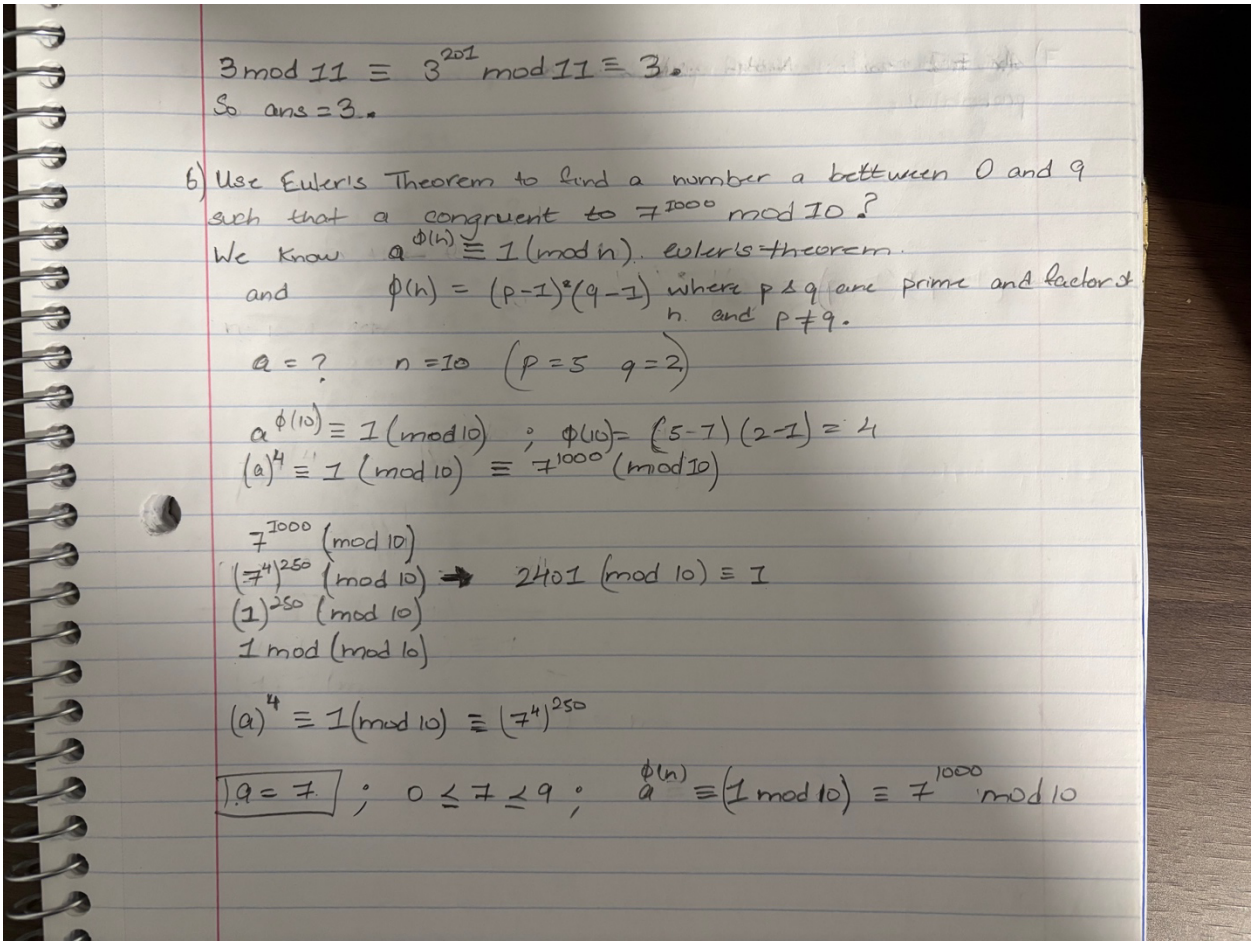
$7^{1000} \pmod{10}$

$(7^4)^{250} \pmod{10}$ → $2401 \pmod{10} \equiv 1$

$(1)^{250} \pmod{10}$

$1 \bmod \pmod{10}$

Q6: Use Euler's Theorem to find a number a between 0 and 9 such that a is congruent to 71000 mod 10.



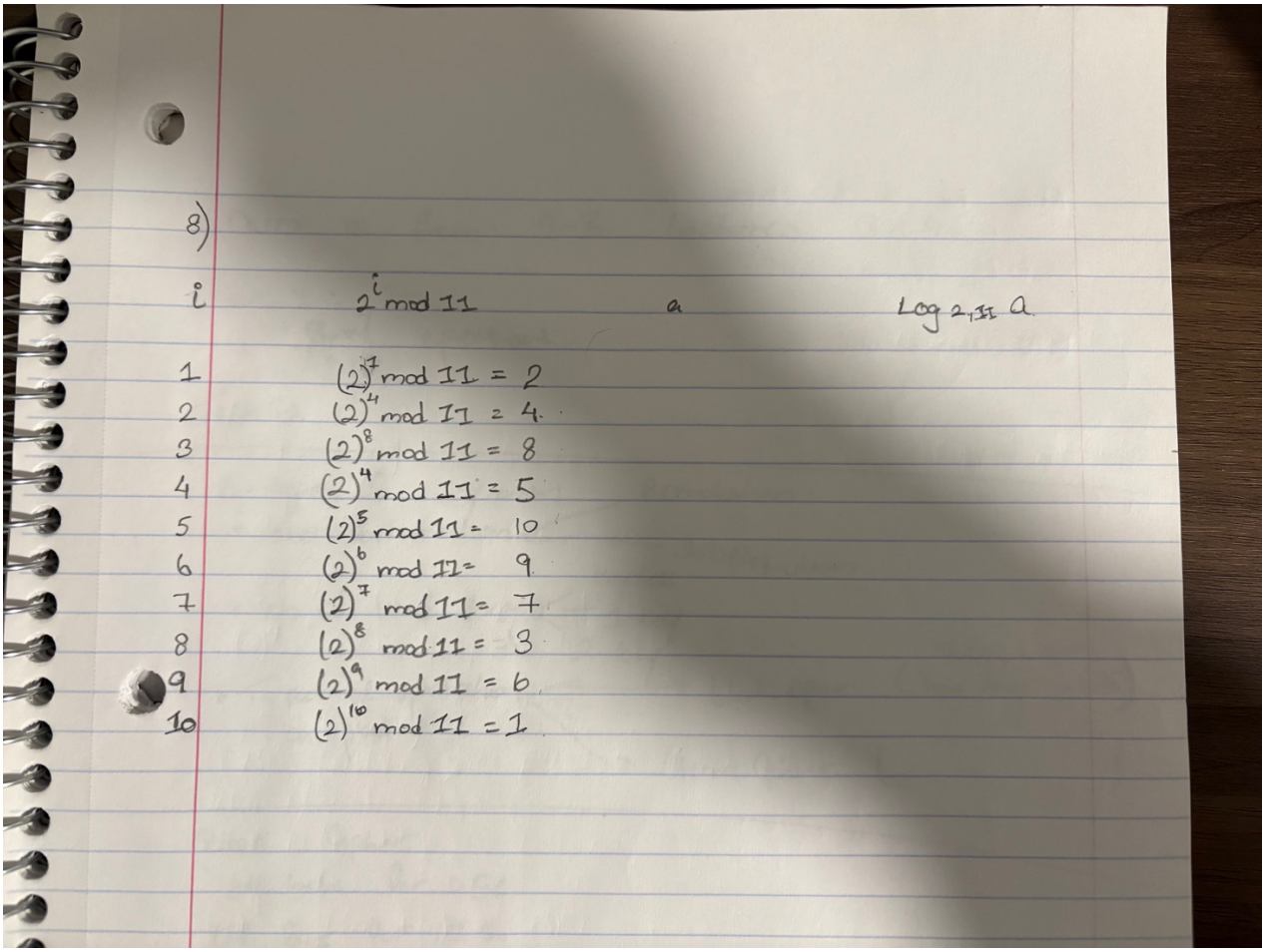
Q7: The Miller-Rabin test can determine if a number is not prime but cannot determine if a number is prime. How can such an algorithm be used to test for primality?

The Miller-Rabin test is a probabilistic algorithm that can determine if a number is composite but cannot definitively prove that a number is prime. However, it can still be effectively used to test for primality.

While the test is not deterministic, because its probabilistic it is acceptable. The likelihood of a number being prime increases significantly with repeated independent runs. By running the

Miller-Rabin test multiple times with different bases, we can reduce the probability of a false positive with a high level of confidence. Additionally, the test guarantees no false negatives, so if a number is not prime, the algorithm will always detect it as a composite.

Q8: Given 2 as a primitive root of 11 construct a table of discrete logarithms.



Resources Used:

https://www.youtube.com/watch?v=JoeiLuFNBc4&list=PLBlnK6fEyqRhBsP45jUdcqBivf25hyVkU

Task 2: Extended Euclidean Algorithm Implementation

Output Screenshots:

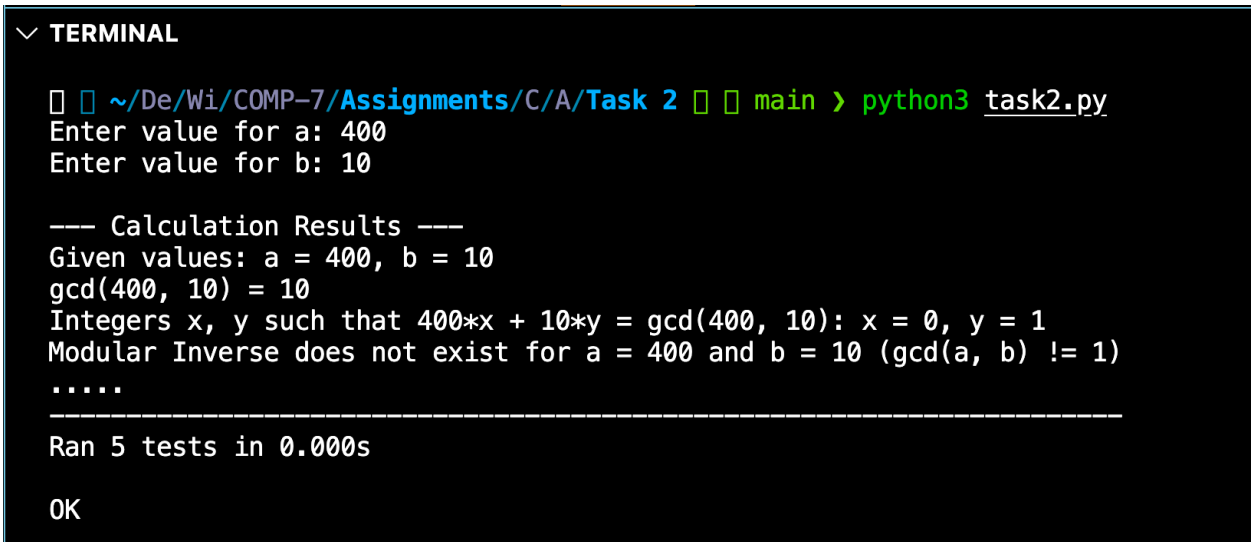
Test Cases:

```
class TestExtendedEuclideanAlgorithm(unittest.TestCase):

    def testcase_1(self):
        a, b = 43, 17
        gcd, x, y = egcd(a, b)    "egcd": Unknown word.
        self.assertEqual(gcd, 1)
        self.assertEqual(a * y + b * x, gcd)
        result = modInv(a, b)
        self.assertEqual(result, 38)

    def testcase_2(self):
        a, b = 12, 8
        gcd, x, y = egcd(a, b)    "egcd": Unknown word.
        self.assertEqual(gcd, 4)
        self.assertEqual(a * y + b * x, gcd)
        result = modInv(a, b)
        self.assertIsNone(result)

    def testcase_3(self):
        a, b = 43, 1
        gcd, x, y = egcd(a, b)    "egcd": Unknown word.
        self.assertEqual(gcd, 1)
        self.assertEqual(a * y + b * x, gcd)
        result = modInv(a, b)
        self.assertEqual(result, 1)
```

```python
class TestExtendedEuclideanAlgorithm(unittest.TestCase):

    def testcase_4(self):
        a, b = 123456789, 98765432
        gcd, x, y = egcd(a, b)    "egcd": Unknown word.
        self.assertEqual(gcd, 1)
        self.assertEqual(a * y + b * x, gcd)
        result = modInv(a, b)
        self.assertEqual(result, 92592593)

    def testcase_5(self):
        a, b = 25, 25
        gcd, x, y = egcd(a, b)    "egcd": Unknown word.
        self.assertEqual(gcd, 25)
        self.assertEqual(a * y + b * x, gcd)
        result = modInv(a, b)
        self.assertIsNone(result)
```

Output:

```
 ~/De/Wi/COMP-7/Assignments/C/A/Task 2  main !1 ❯ python3 task2.py
Enter value for a: 43
Enter value for b: 17

--- Calculation Results ---
Given values: a = 43, b = 17
gcd(43, 17) = 1
Integers x, y such that 43*x + 17*y = gcd(43, 17): x = 2, y = -5
Modular Inverse of 43 modulo 17 is: 38
.....
----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
```



Resources Used:

https://www.youtube.com/watch?v=JoeiLuFNBc4&list=PLBlnK6fEyqRhBsP45jUdcqBivf25hyVkU

Task 3: Implementing the Playfair Cipher

Output Screenshots:

Test Cases:

```
Aryan Jand, 42 seconds ago | 1 author (Aryan Jand)
class TestPlayfairCipher(unittest.TestCase):    "Playfair": Unknown word.

    def testcase_1(self):
        key = ''
        message = ''
        res = playfair_cipher(key, message)    "playfair": Unknown word.
        expected_output = ''
        self.assertEqual(res, expected_output)

    def testcase_2(self):
        key = ''
        message = 'hello'
        res = playfair_cipher(key, message)    "playfair": Unknown word.
        expected_output = 'KCNVMP'      "KCNVMP": Unknown word.
        self.assertEqual(res, expected_output)

    def testcase_3(self):
        key = 'ballon'
        message = 'world'
        res = playfair_cipher(key, message)    "playfair": Unknown word.
        expected_output = 'YASAEW'    "YASAEW": Unknown word.
        self.assertEqual(res, expected_output)

    def testcase_4(self):
        key = 'MONARCHY monarchy'
        message = 'mosque'
        res = playfair_cipher(key, message)    "playfair": Unknown word.
        expected_output = 'ONTSML'      "ONTSML": Unknown word.
        self.assertEqual(res, expected_output)
```

```
class TestPlayfairCipher(unittest.TestCase):    "Playfair": Unknown word.
    def testcase_5(self):
        key = 'kingdom'
        message = 'hello world'
        res = playfair_cipher(key, message)    "playfair": Unknown word.
        expected_output = 'LFHYEBVMTFNZ'      "LFHYEBVMTFNZ": Unknown word.
        self.assertEqual(res, expected_output)

    def testcase_6(self):
        key = 'playfair'    "playfair": Unknown word.
        message = 'jjj'
        res = playfair_cipher(key, message)    "playfair": Unknown word.
        expected_output = 'CUCUCU'      "CUCUCU": Unknown word.
        self.assertEqual(res, expected_output)

    def testcase_7(self):
        key = None
        message = 'message'
        res = playfair_cipher(key, message)    "playfair": Unknown word.
        self.assertIsNone(res)

    def testcase_8(self):
        key = 'keyword'
        message = None
        res = playfair_cipher(key, message)      "playfair": Unknown word.
        self.assertIsNone(res)
```

Output:

```
  ~/De/Wi/COMP-7/Assignments/C/A/Task 3    main !1 ❯ python3 task3.py
Enter the key for the Playfair cipher: hello
Enter the message to encrypt: hello world
Encrypted message:
ELDLOAYESEML
........
----------------------------------------------------------------------
Ran 8 tests in 0.001s

OK
```

Resources Used:

https://www.youtube.com/watch?v=JoeiLuFNBc4&list=PLBlnK6fEyqRhBsP45jUdcqBivf25hyVkU