Question 1)

PART A) Response from the server:

```
{'n': 419, 't': 11}
Returned values n and t: 419, 11
```

Firstly, I checked if the n = 419 is prime or not by using SymPy library:

```
Is my 'n = 419' prime number?: True
```

Therefore, we can say that we have "n-1" elements in our group as n is a prime number. In my case, we have 418 elements in the group [1, 418]. We can see the server response: **Congrats!**

```
# Question 1 Part A
n, t = getQ1()
print("Returned values n and t: {}, {}".format(n,t))
print("Is my 'n = {}' prime number?: {}".format(n, sympy.isprime(n)))
checkQ1a(418)
```

Output:

```
{'n': 419, 't': 11}
Returned values n and t: 419, 11
Is my 'n = 419' prime number?: True
Congrats!
```

PART B) I used following algorithm to find generators of my group:

```
my_group = set(range(1, n))
generators = []
for i in range(1, n):
    my_list = set({})
    for k in range(1, n):
        val = (i ** k) % n
        my_list.add(val)
    if my_list == my_group:
        generators.append(i)
```

In the outer loop I iterated over all my group elements. In the inner loop, for each element I took all powers of my element and took their modulo n and add the result corresponding set for each element. At the end, if element's list equal to my group, that element is a generator of my group.

According to lecture notes, If modulo p is a prime, we have phi(p-1) number of generators for the group. I checked my algorithm results to see if it is consistent. Yes, it is consistent. We need to have phi(418) = 180 generators, and I found 180 generators with my algorithm. I printed 5 examples of generators and send 2 of them to the server and got the result **"Congrats!".**

```
Number of generators should be 180 according to phi function of n-1 = 418
Number of generators of my group according to my algorithm: 180
Smallest 5 generators of my group: [2, 6, 8, 10, 11]
Check Question 1b response from server
Given generator is -> 2, corresponding response from server -> Congrats!
Given generator is -> 6, corresponding response from server -> Congrats!
All of the generators:  [2, 6, 8, 10, 11, 14, 17, 18, 19, 24, 26, 30, 31, 32, 33, 42, 44, 46, 50, 51, 53, 54, 55, 56,
57, 58, 61, 67, 68, 70, 72, 74, 77, 78, 82, 83, 86, 93, 94, 95, 96, 98, 99, 101, 103, 104, 109, 118, 120, 124, 126,
127, 128, 130, 132, 133, 138, 143, 146, 150, 153, 155, 158, 159, 160, 162, 163, 165, 167, 168, 174, 176, 179, 181,
182, 183, 184, 193, 194, 200, 201, 210, 212, 213, 214, 216, 217, 221, 222, 223, 224, 227, 228, 229, 230, 231, 232,
233, 234, 239, 241, 242, 244, 246, 247, 249, 253, 255, 258, 262, 263, 265, 268, 270, 271, 272, 274, 275, 277, 278,
279, 282, 285, 288, 294, 296, 297, 298, 302, 303, 304, 307, 308, 309, 311, 313, 314, 319, 322, 327, 328, 331, 332,
335, 338, 339, 340, 344, 346, 353, 354, 355, 356, 357, 367, 371, 374, 376, 378, 380, 381, 382, 383, 384, 385, 390,
391, 392, 394, 396, 397, 398, 399, 403, 404, 407, 410, 414, 415, 416]
```

PART C)

We have to find a subgroup with size t (in my case t = 11). Since t divides the order of original group. It is a valid order. In my case, my big group's order is 418, and t = 11.

I found this subgroup with the following algorithm:

```python
for i in range(1,n):
    my_list = set({})
    for k in range(1,n):
        val = (i ** k) % n
        my_list.add(val)
    if len(my_list) == t:
        print("\nThis is the order t = {} subgroup: {}".format(t, my_list))
        break
```

Output of above algorithm:

```
This is the order t = 11 subgroup: {129, 1, 69, 102, 169, 300, 13, 334, 152, 59, 348}
```

Then, I found the generators of my subgroup with the following algorithm:

```python
my_subgroup = my_list
generators = []
for i in my_list:
    temp = set({})
    for k in range(1,n):
        val = (i ** k) % n
        temp.add(val)
    if temp == my_subgroup:
        generators.append(i)
print("Generators of my sub group: {}".format(generators))
```

Output of above algorithm:

```
Generators of my sub group: [129, 69, 102, 169, 300, 13, 334, 152, 59, 348]
```

These are the generators of my sub-group. Then I send one of them to the server and I got the

Congrats! Response from the server.

```
This is the order t = 11 subgroup: {129, 1, 69, 102, 169, 300, 13, 334, 152, 59, 348}
Generators of my sub group: [129, 69, 102, 169, 300, 13, 334, 152, 59, 348]
Congrats!
```

Question 2)

Firstly, I got my e and cipher values from the server. Their values are below:

```
Returned values of e:
7919205075575724789805185337017354112760749011078016506530371000270943138879349429740869227729970132245368919004768491
8581100088273774845605271266817804705615922287972311611338084160196475521074474048261324497616428721398148750661253559
4423050693665989124576341262477450239638015632008839587442754168251787279
cipher:
6582593786609979258309136697934407749383133664058524349597692142376280469351349100625445299706410469543784801758453221
9136684089856445633679479907332559210371045549851487757734233143499393878173109335946780447877631429414072228566115349
2795851429870227141924440322063765318471434761178415130027495325921615804
```

I need to calculate phi of n to calculate d. We know that phi(n) can be written as following way:

$$\text{If } n = p \cdot q \text{ then } \phi(n) = (p-1)\cdot(q-1)$$

```
Our phi of n is:
1475935655289459665540345368401426056707076644459891805389807445048663539847240897383283456087916140664695038967370287
4237237178460658738643689092285671479314778725021686293591872879232296500947715590121557363217428397910825651704761376
79638968866434631195957139436914604721781841989914765163879639073172804008
```

Then, I calculated my d value by using given function modinv.

```
Our d is:
4273237425586206818411831744174632447689297216458903751218549756871027775852414075371029294360882284256245961762865983
3874927277153361509652701418816702396677836164732856808536253109913014337727394288790514751127004610543123072903110376
9916722350683130840469875189507874378305103117599438451594337766149407175
```

Rest is easy as we have all the values to calculate m:

m = pow(cipher, d, n) → d is the power, n is the modulo

If we try to do it like (cipher ** d) % n, it takes too much time.

```
Our m is:
1409754382171986085528226313433989527792792785127086468413498824955659791387686055991775396242000938736285941546586431
075797580151821938012249869737149106349694389190598466196672942790630705030995801279946083553065017
```

Then, we need to convert this m to bytes. Then, we need to decode it for obtaining plain text.

```
57      n = p * q
58      phi_n = (p-1) * (q-1)
59      d = modinv(e, phi_n)
60      m = pow(cipher, d, n)
61
62      print("Our phi of n is: {}\nOur d is: {}\nOur m is: {}".format(phi_n, d, m))
63
64      # total_length = (m.bit_length() // 8) + 1
65      my_bytes = m.to_bytes(m.bit_length(), byteorder='big')
66      my_bytes = my_bytes.lstrip(b'\x00')
67      text = my_bytes.decode('utf-8')
```

To_bytes function's first parameter is the length of resulted array. Since we are using utf-8, I could calculate the length of the resulted array as line 64 (+1 is the escape character). However, I just give the bit length of m as the length of the resulted array which is larger than actual result. Then, I stripped the escape characters from left side to obtain my plain text.

```
Our text is: Answer to the ultimate question of life, the universe, and everything is not 42. it is 589
Congrats!
```

We can see the plain text and corresponding server response for this text in the above.

Question 3)

We know that one of the cipher texts is uncorrupted. Therefore, we need to find it.

I tried to decrypt all 3 cipher texts. Only Cipher text 2 is able to decrypt without problem. Since we have used same nonce for each message, we can use cipher text 2's nonce for the others. Our nonce can be seen below:

```
nonce:  b'\x9d\x131v-\xdda\xe9'
```

Then, I started to try other cipher texts to decrypt with this nonce. Since we know some of the bytes of nonces are missing, I did exhaustive search to identify message parts of the cipher texts as following way:

```python
message1 = []
for i in range(8):
    ciphertext = cipher_text1[i:]
    cipher = Salsa20.new(key, nonce=ctext_nonce2)
    dtext = cipher.decrypt(ciphertext)
    message1.append(dtext.decode('UTF-8',errors='ignore'))
```

After this step, I had up to 8 different plain texts (some of them was not able to decoded). Then, I could choose with observing output of messages list. However, I used NLTK tool like I did in the Homework 1 to find English sentence in the list. **Results as following:**

Message 1: I love deadlines. I love the whooshing noise they make as they go by

Message 2: Our knowledge can only be finite, while our ignorance must necessarily be infinite

Message 3: Any unwillingness to learn mathematics today can greatly restrict your possibilities tomorrow

```
Plain text 1:  I love deadlines. I love the whooshing noise they make as they go by
Plain text 2:  Our knowledge can only be finite, while our ignorance must necessarily be infinite
Plain text 3:  Any unwillingness to learn mathematics today can greatly restrict your possibilities tomorrow
```

Question 4)

We know that if gcd(a, n) = 1, we have exactly one solution and we can calculate it easily.

If the gcd(a, n) = d != 1, we may have d number of solutions or we may not have a solution for this equation. If **d does not divide b**, we do not have any solutions. Otherwise, we have d many solutions. According to this knowledge, I made an algorithm as below:

```
def solve(n, a, b):
  my_gcd = gcd(a, n)
  print("Gcd of a and n is: ", my_gcd)
  if (my_gcd == 1):
      print("There is exactly one solution!")
      x = (modinv(a, n) * b) % n
      return x
  else:
      if (b % my_gcd) == 0:
          print("There are {} solutions!".format(my_gcd))
          results = []
          new_a = a // my_gcd
          new_b = b // my_gcd
          new_n = n // my_gcd
          x = (modinv(new_a, new_n) * new_b) % new_n
          for i in range(my_gcd):
              x_ = x + (i * new_n)
              results.append(x_)
          return results
      else:
          print("{} does not divide {}".format(my_gcd, b))
          return "SOLUTION DOES NOT EXIST"
```

Part A)

Result: 56884393062303769019751445983612369117060043083722821988604

```
Gcd of a and n is:  1
There is exactly one solution!
Result: 56884393062303769019751445983612369117060043083722821988604
```

Part B)

Result: No solution

```
Gcd of a and n is:  3
3 does not divide 1267565499436628521023818343520287296453722217373643204657115
Result: SOLUTION DOES NOT EXIST
```

Since our gcd = 3 does not divide the ‘b’, we do not have any solutions for this equation.

Part C)

```
Gcd of a and n is:  3
There are 3 solutions!
Result: [9609279374756105288427021898499890361717105145551739027963,
110042907140942997509799652683766709621865315673440026493694,
2104765349071298897311722834690335288820135262013283139594 25]
```

We found all results according to this formula:

$$x = \left\{ \tilde{x}, \tilde{x} + \frac{n}{d}, \tilde{x} + 2\frac{n}{d}, \cdots, \tilde{x} + (d-1)\frac{n}{d} \right\}$$

We are just adding the new modulo to our base result.

Question 5)

We know that binary connection polynomial for LFSR can have maximum period of 2^L-1.

This means that our register value come back to its initial state after this period. Therefore, we can check register states to observe periodicity. I made an algorithm for this purpose:

```python
def shift(pol, new_bit):
    temp = []
    temp.append(new_bit)
    for i in range(len(pol)-1):
        temp.append(pol[i])
    return temp



# Polynom 1 -> x^5 + x^2 + 1

S1 = [0,0,0,0,1]
initial = S1.copy()

print("Initial state: {}".format(initial))
for i in range(2**len(S1)-1):
    S1 = shift(S1, S1[1]^S1[4])
    print("{}. state: {}".format(i+1, S1))
    if S1 == initial:
        print("We completed our cycle in {} states".format(i+1))
        break
if i+1 == 2**len(S1)-1:
    print("Connection polynomial for LFSR produces maximum period sequence!")
else:
    print("It does not produce maximum period sequence!")
```

It is basically shifting our register and checking if we reach our initial state or not. At the end, we are checking if we reached initial state in how many steps.

For the polynomial 1: $p_1(x) = x^5 + x^2 + 1$

Expected Maximum period sequence: $2^5-1 = 31$

Therefore, we are expecting to reach initial state at the 31^{st} state.

```
Initial state: [0, 0, 0, 0, 1]
1. state: [1, 0, 0, 0, 0]
2. state: [0, 1, 0, 0, 0]
3. state: [1, 0, 1, 0, 0]
4. state: [0, 1, 0, 1, 0]
5. state: [1, 0, 1, 0, 1]
6. state: [1, 1, 0, 1, 0]
7. state: [1, 1, 1, 0, 1]
8. state: [0, 1, 1, 1, 0]
9. state: [1, 0, 1, 1, 1]
10. state: [1, 1, 0, 1, 1]
11. state: [0, 1, 1, 0, 1]
12. state: [0, 0, 1, 1, 0]
13. state: [0, 0, 0, 1, 1]
14. state: [1, 0, 0, 0, 1]
15. state: [1, 1, 0, 0, 0]
16. state: [1, 1, 1, 0, 0]
17. state: [1, 1, 1, 1, 0]
18. state: [1, 1, 1, 1, 1]
19. state: [0, 1, 1, 1, 1]
20. state: [0, 0, 1, 1, 1]
21. state: [1, 0, 0, 1, 1]
22. state: [1, 1, 0, 0, 1]
23. state: [0, 1, 1, 0, 0]
24. state: [1, 0, 1, 1, 0]
25. state: [0, 1, 0, 1, 1]
26. state: [0, 0, 1, 0, 1]
27. state: [1, 0, 0, 1, 0]
28. state: [0, 1, 0, 0, 1]
29. state: [0, 0, 1, 0, 0]
30. state: [0, 0, 0, 1, 0]
31. state: [0, 0, 0, 0, 1]
We completed our cycle in 31 states
Connection polynomial for LFSR produces maximum period sequence!
```

We reached at the 31. State. Therefore, **Polynomial 1 generates maximum period sequence!**

For the polynomial 2: $p_2(x) = x^5 + x^3 + x^2 + 1$

Expected Maximum period sequence: $2^5-1 = 31$

```
Initial state: [0, 0, 0, 0, 1]
1. state: [1, 0, 0, 0, 0]
2. state: [0, 1, 0, 0, 0]
3. state: [1, 0, 1, 0, 0]
4. state: [1, 1, 0, 1, 0]
5. state: [1, 1, 1, 0, 1]
6. state: [1, 1, 1, 1, 0]
7. state: [0, 1, 1, 1, 1]
8. state: [1, 0, 1, 1, 1]
9. state: [0, 1, 0, 1, 1]
10. state: [0, 0, 1, 0, 1]
11. state: [0, 0, 0, 1, 0]
12. state: [0, 0, 0, 0, 1]
We completed our cycle in 12 states
It does not produce maximum period sequence!
```

We reached at the 12. State. Therefore, **Polynomial 2 does not generate maximum period sequence!**

Question 6)

We have seen that the expected linear complexity of random sequence is approximately half of its length. (Constant does not important)

$$E(L(s^n)) \approx n/2 + \cancel{2/9}.$$

Therefore, when we give our sequences to the Berlekamp-Massey Algorithm (BMA) which is provided in the homework, we can see the linear complexity of sequences. If the result is around n/2, we can say they are unpredictable, random sequences.

```
Length of the sequence x1:  84
Expected Linear Complexity for Unpredictable Sequence -->  42.0
Linear Complexity of our sequence x1 according to Berlekamp-Massey Algorithm (BMA):  31

Length of the sequence x2:  90
Expected Linear Complexity for Unpredictable Sequence -->  45.0
Linear Complexity of our sequence x2 according to Berlekamp-Massey Algorithm (BMA):  31

Length of the sequence x3:  89
Expected Linear Complexity for Unpredictable Sequence -->  44.5
Linear Complexity of our sequence x3 according to Berlekamp-Massey Algorithm (BMA):  31
```

We can see that all three sequences have Linear Complexity of 31. This result is lower than expected linear complexities. Therefore, these three sequences are **not unpredictable.**

Question 7)

We know that message ends with “Erkay Savas”. Therefore, we need to convert it to the binned format. Then, when we XOR this binned “Erkay Savas” with the corresponding cipher text part, we will obtain the last part of the key.

```
Binned version of Erkay Savas and its size: [1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0,
0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0,
0, 0, 1, 1, 1, 1, 0, 0, 1, 1], 77

Corresponding Cipher Text and its size: [0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0,
1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0,
0, 0, 1, 0, 1, 1, 1, 1, 0], 77

Known part of the key with length '77' is: [1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1,
0, 0, 1, 0, 1, 0, 1, 1, 0, 1]
```

Then, we can use Berlekamp-Massey Algorithm (BMA) to find key’s Linear Complexity and Connection Polynomial if the known part of the key is sufficiently long.

Linear Complexity: 26

Connection Polynomial: [1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

```
Linear Complexity '26', and Connection Polynomial according to Berlekamp-Massey Algorithm (BMA) is: [1, 1, 1, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
```

Now, we need to find initial state of the key to find whole key. My first thinking was doing an exhaustive search for the initial state. However, we have 2^26 possibilities for the initial state which would need large computational power to try all possibilities. Then, I decided to make a different approach. Since we know the only last part of the key, I tried to find the key in a reversed manner. In this case, our initial state was the last 26 bits of the known key as its reversed. Since we are going backward, we need also backward Connection polynomial. Then, we can find the whole reversed key with this initial state and connection polynomial. At the end, we need to reverse the key again to obtain original key. After all these operations, I obtained the following plain text:

Plain text: Dear Student,

You have worked hard, I know taht; but it paid off:)

You have just earned 20 points.

Congrats!

Best, Erkay Savas

```
Plain text:
Dear Student,
You have worked hard, I know taht; but it paid off:)
You have just earned 20 points.
Congrats!
Best, Erkay Savas
```

Ege Erdoğan 25331 (*ALL CODES AT THE END OF PDF*)

CODES

Question 1

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 31 12:57:41 2021

@author: user
"""

import math
import warnings
import sympy
import random
import requests

#API_URL = 'http://10.36.52.109:6000'
API_URL = 'http://cryptlygos.pythonanywhere.com'

my_id = 25331

def getQ1():
  endpoint = '{}/{}/{}'.format(API_URL, "Q1", my_id )
  response = requests.get(endpoint)
  if response.ok:
    res = response.json()
    print(res)
    n, t = res['n'], res['t']
    return n,t
  else: print(response.json())


def checkQ1a(order):   #check your answer for Question 1 part a
  endpoint = '{}/{}/{}/{}'.format(API_URL, "checkQ1a", my_id, order)
  response = requests.put(endpoint)
  print(response.json())


def checkQ1b(g):  #check your answer for Question 1 part b
  endpoint = '{}/{}/{}/{}'.format(API_URL, "checkQ1b", my_id, g)    #gH is generator of your subgroup
  response = requests.put(endpoint)     #check result
  #print(response.json())
  return response.json()


def checkQ1c(gH):  #check your answer for Question 1 part c
  endpoint = '{}/{}/{}/{}'.format(API_URL, "checkQ1c", my_id, gH )  #gH is generator of your subgroup
  response = requests.put(endpoint)     #check result
  print(response.json())

def phi(n):
    amount = 0
    for k in range(1, n + 1):
        if math.gcd(n, k) == 1:
            amount += 1
    return amount
```

Ege Erdoğan 25331 (*ALL CODES AT THE END OF PDF*)

```python
def gcd(a, b):
    """Calculate the Greatest Common Divisor of a and b.

    Unless b==0, the result will have the same sign as b (so that when
    b is divided by it, the result comes out positive).
    """
    while b:
        a, b = b, a%b
    return a

# # Question 1 Part A
n, t = getQ1()
print("Returned values n and t: {}, {}".format(n,t))
print("Is my 'n = {}' prime number?: {}".format(n, sympy.isprime(n)))
checkQ1a(418)

# Question 1 Part B
my_group = set(range(1, n))
generators = []
for i in range(1, n):
    my_list = set({})
    for k in range(1, n):
        val = (i ** k) % n
        my_list.add(val)
    if my_list == my_group:
        generators.append(i)

print("\nNumber of generators should be {} according to phi function of n-1 = {}".format(phi(n-1), n-1))
print("Number of generators of my group according to my algorithm: {}".format(len(generators)))
print("Smallest 5 generators of my group: {}".format(generators[:5]))
print("Check Question 1b response from server")
print("Given generator is -> {}, corresponding response from server -> {}".format(generators[0], checkQ1b(generators[0])))
print("Given generator is -> {}, corresponding response from server -> {}".format(generators[1], checkQ1b(generators[1])))
print("All of the generators: ", generators)

# Question 1 Part C
for i in range(1,n):
    my_list = set({})
    for k in range(1,n):
        val = (i ** k) % n
        my_list.add(val)
    if len(my_list) == t:
        print("\nThis is the order t = {} subgroup: {}".format(t, my_list))
        break

my_subgroup = my_list
generators = []
for i in my_list:
    temp = set({})
    for k in range(1,n):
        val = (i ** k) % n
        temp.add(val)
    if temp == my_subgroup:
        generators.append(i)
print("Generators of my sub group: {}".format(generators))
checkQ1c(300)
```

Ege Erdoğan 25331 (*ALL CODES AT THE END OF PDF*)

Question 2

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 31 14:37:40 2021

@author: user
"""
import random
import requests
import math
import warnings
import sympy
#API_URL = 'http://10.36.52.109:6000'
API_URL = 'http://cryptlygos.pythonanywhere.com'
my_id = 25331
def getQ2():
  endpoint = '{}/{}/{}'.format(API_URL, "Q2", my_id )
  response = requests.get(endpoint)
  if response.ok:
    res = response.json()
    e, cipher = res['e'], res['cipher']
    return e, cipher
  else:  print(response.json())
def checkQ2(ptext):  #check your answer for Question 1 part c
  response = requests.put('{}/{}'.format(API_URL, "checkQ2"), json = {"ID": my_id, "msg":ptext})
  print(response.json())
def egcd(a, b):
    x,y, u,v = 0,1, 1,0
    while a != 0:
        q, r = b//a, b%a
        m, n = x-u*q, y-v*q
        b,a, x,y, u,v = a,r, u,v, m,n
    gcd = b
    return gcd, x, y

def modinv(a, m):
    gcd, x, y = egcd(a, m)
    if gcd != 1:
        return None  # modular inverse does not exist
    else:
        return x % m
p = 23736540918088479407817876031701066644301064882958875296167214819014438374011661672830210955539507252066999384067356159056835877
q = 62179896404564992443617709894241054520624355558658288422696178839274611833136662241430162694076231401545584449128278988404970580
e, cipher = getQ2()
print("Returned values of e: {}\ncipher: {}".format(e,cipher))
n = p * q
phi_n = (p-1) * (q-1)
d = modinv(e, phi_n)
m = pow(cipher, d, n)
print("Our phi of n is: {}\nOur d is: {}\nOur m is: {}".format(phi_n, d, m))
# total_length = (m.bit_length() // 8) + 1
my_bytes = m.to_bytes(m.bit_length(), byteorder='big')
my_bytes = my_bytes.lstrip(b'\x00')
text = my_bytes.decode('utf-8')
print("Our text is: {}".format(text))
checkQ2(text)
```

Ege Erdoğan 25331 (*ALL CODES AT THE END OF PDF*)

Question 3

```python
"""
#!pip install pycryptodome
from Crypto.Cipher import Salsa20
import random
import nltk
nltk.download('words')
from nltk.corpus import words

cipher_text1 = b'1v-\xdda\x9d\x13\xf5y\xd4M\xcc\xc2\xd5\xc9\xe8\xca\xfcF\xe1\x7f\xdd\xabM,=c\xa6\x9e\xd2M\x11;9Bpna\x91\xb8\xf5z>\x0c
cipher_text2 = b'\x9d\x131v-\xdda\xe9\xf3,\xca\x02\xd1\xc9\x9a\xda\xe1\xce\xfcM\xed1\xdb\xb9\r,\x1b-\xa6\x88\x84JTo7N>p}\x9b\xfb\xa6e
cipher_text3 = b"\x00\x04\x00\x00\x00\x00\xfd7\xc1\x02\xcf\xc9\x82\xc4\xe1\xc7\xf1D\xef\x7f\xdd\xab\x10,\x00,\xea\x9d\xc1IC!qJ1ma\x9b
secret = 314159265358979323
key = secret.to_bytes(32, byteorder='big')
print("Key length in bytes", len(key))

unbroken_msg = []
ctext_nonce1 = cipher_text1[:8]
ciphertext1 = cipher_text1[8:]
cipher = Salsa20.new(key, nonce=ctext_nonce1)
dtext1 = cipher.decrypt(ciphertext1)
unbroken_msg.append(dtext1.decode('UTF-8',errors='ignore'))
ctext_nonce2 = cipher_text2[:8]
ciphertext2 = cipher_text2[8:]
cipher = Salsa20.new(key, nonce=ctext_nonce2)
dtext2 = cipher.decrypt(ciphertext2)
unbroken_msg.append(dtext2.decode('UTF-8',errors='ignore'))
ctext_nonce3 = cipher_text3[:8]
ciphertext3 = cipher_text3[8:]
cipher = Salsa20.new(key, nonce=ctext_nonce3)
dtext3 = cipher.decrypt(ciphertext3)
unbroken_msg.append(dtext3.decode('UTF-8',errors='ignore'))
print("nonce: ", ctext_nonce2)
print("Unbroken message: ", unbroken_msg[1])
message1 = []
for i in range(8):
    ciphertext = cipher_text1[i:]
    cipher = Salsa20.new(key, nonce=ctext_nonce2)
    dtext = cipher.decrypt(ciphertext)
    message1.append(dtext.decode('UTF-8',errors='ignore'))
message3 = []
for i in range(8):
    ciphertext = cipher_text3[i:]
    cipher = Salsa20.new(key, nonce=ctext_nonce2)
    dtext = cipher.decrypt(ciphertext)
    message3.append(dtext.decode('UTF-8',errors='ignore'))
for i in range(len(message1)):
    temp = message1[i].split()
    if temp[0].lower() in words.words():
        print("\nPlain text 1: ", message1[i])
print("Plain text 2: ", unbroken_msg[1])
for i in range(len(message3)):
    temp = message3[i].split()
    if temp[0].lower() in words.words():
        print("Plain text 3: ", message3[i])
```

Ege Erdoğan 25331 (*ALL CODES AT THE END OF PDF*)

Question 4

```python
def modinv(a, m):
    gcd, x, y = egcd(a, m)
    if gcd != 1:
        return None  # modular inverse does not exist
    else:
        return x % m
def solve(n, a, b):
  my_gcd = gcd(a, n)
  print("Gcd of a and n is: ", my_gcd)
  if (my_gcd == 1):
      print("There is exactly one solution!")
      x = (modinv(a, n) * b) % n
      return x
  else:
      if (b % my_gcd) == 0:
          print("There are {} solutions!".format(my_gcd))
          results = []
          new_a = a // my_gcd
          new_b = b // my_gcd
          new_n = n // my_gcd
          x = (modinv(new_a, new_n) * new_b) % new_n
          for i in range(my_gcd):
              x_ = x + (i * new_n)
              results.append(x_)
          return results
      else:
          print("{} does not divide {}".format(my_gcd, b))
          return "SOLUTION DOES NOT EXIST"
#Part A
n = 100433627766186892221372630785266819260148210527888287465731
a = 336819975970284283819362806770432444188296307667557062083973
b = 25245096981323746816663608120290190011570612722965465081317
result = solve(n,a,b)
print("Result: {}\n".format(result))
#Part B
n = 301300883298560676664117892355800457780444631583664862397193
a = 1070400563622371146605725585064882995936005838597136294785034
b = 1267565499436628521023818343520287296453722217373643204657115
result = solve(n,a,b)
print("Result: {}\n".format(result))
#Part C
n = 301300883298560676664117892355800457780444631583664862397193
a = 608240182465796871639779713869214713721438443863110678327134
b = 721959177061605729962797351110052890685661147676448969745292
result = solve(n,a,b)
print("Result: {}\n".format(result))
```

Question 5

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  1 15:52:12 2021

@author: user
"""



def shift(pol, new_bit):
    temp = []
    temp.append(new_bit)
    for i in range(len(pol)-1):
        temp.append(pol[i])
    return temp

# Polynom 1 -> x^5 + x^2 + 1

S1 = [0,0,0,0,1]
initial = S1.copy()
print("Initial state: {}".format(initial))
for i in range(2**len(S1)-1):
    S1 = shift(S1, S1[1]^S1[4])
    print("{}. state: {}".format(i+1, S1))
    if S1 == initial:
        print("We completed our cycle in {} states".format(i+1))
        break
if i+1 == 2**len(S1)-1:
    print("Connection polynomial for LFSR produces maximum period sequence!")
else:
    print("It does not produce maximum period sequence!")


# Polynom 2 -> x^5 + x^3 +x^2 + 1

S1 = [0,0,0,0,1]
initial = S1.copy()

print("\nInitial state: {}".format(initial))
for i in range(2**len(S1)-1):
    S1 = shift(S1, S1[1]^S1[4]^S1[2])
    print("{}. state: {}".format(i+1, S1))
    if S1 == initial:
        print("We completed our cycle in {} states".format(i+1))
        break
if i+1 == 2**len(S1)-1:
    print("Connection polynomial for LFSR produces maximum period sequence!")
else:
    print("It does not produce maximum period sequence!")
```

Ege Erdoğan 25331 (*ALL CODES AT THE END OF PDF*)

Question 6

```python
def BM(s):
    n = len(s)

    C = []
    B = []
    T = []
    L = 0
    m = -1
    i = 0
    C.append(1)
    B.append(1)

    while(i<n):
        delta = 0
        clen = len(C)
        for j in range(0, clen):
            delta ^= (C[j]*s[i-j])
        if delta == 1:
            dif = i-m
            PolCopy(T, C)
            nlen = len(B)+dif
            if(clen >= nlen):
                for j in range(dif,nlen):
                    C[j] = C[j] ^ B[j-dif]
            else: # increase the degree of C
                for j in range(clen, nlen):
                    C.append(0)
                for j in range(dif, nlen):
                    C[j] = C[j] ^ B[j-dif]
            PolPrune(C)
            if L <= i/2:
                L = i+1-L
                m = i
                PolCopy(B, T)
        i = i+1
    return L, C
x1 = [0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
x2 = [0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0,
x3 = [1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1,

L1,C1 = BM(x1)
print("Length of the sequence x1: ", len(x1))
print("Expected Linear Complexity for Unpredictable Sequence --> ", len(x1) / 2)
print("Linear Complexity of our sequence x1 according to Berlekamp-Massey Algorithm (BMA): ", L1)

L2,C2 = BM(x2)
print("\nLength of the sequence x2: ", len(x2))
print("Expected Linear Complexity for Unpredictable Sequence --> ", len(x2) / 2)
print("Linear Complexity of our sequence x2 according to Berlekamp-Massey Algorithm (BMA): ", L2)

L3,C3 = BM(x3)
print("\nLength of the sequence x3: ", len(x3))
print("Expected Linear Complexity for Unpredictable Sequence --> ", len(x3) / 2)
print("Linear Complexity of our sequence x3 according to Berlekamp-Massey Algorithm (BMA): ", L3)
```

Question 7

```python
ctext = [1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0
text = "Erkay Savas"
binned_text = ASCII2bin(text)
print("\nBinned version of Erkay Savas and its size: {}, {} ".format(binned_text, len(binned_text)))

cipher_text = ctext[-len(binned_text):]
print("\nCorresponding Cipher Text and its size: {}, {} ".format(cipher_text, len(cipher_text)))

part_key = []
for i in range(len(binned_text)):
    val = binned_text[i] ^ cipher_text[i]
    part_key.append(val)
print("\nKnown part of the key with length '{}' is: {}".format(len(part_key), part_key))


L, C = BM(part_key)
print("\nLinear Complexity '{}', and Connection Polynomial according to Berlekamp-Massey Algorithm (BMA) is: {}".format(L, C))

initial_state = part_key[-L:]

reversed_L, reversed_C = BM(part_key[::-1])
reversed_key = []

for i in range(len(ctext)):
    reversed_key.append(LFSR(reversed_C, initial_state))

key = reversed_key[::-1]

plaintext = []
for i in range(len(ctext)):
    plaintext.append(key[i] ^ ctext[i])
print("\nPlain text: \n{}".format(bin2ASCII(plaintext)))
```