GAYATHRI K

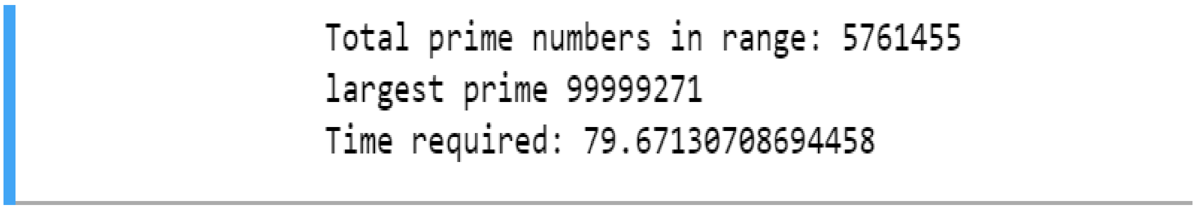
CB.EN.P2CSE19008

BIG DATA SECURITY

1. Code for Finding a large prime number (min 20-digit).

```
import time
def SieveOfEratosthenes(n):
    prime = [True for i in range(n+1)]
    p = 2
    while(p * p <= n):
        if (prime[p] == True):
            for i in range(p * p, n + 1, p):
                prime[i] = False
        p += 1
        c = 0
    for p in range(2, n):
        if prime[p]:
            c += 1
    return c,i
t0 = time.time()
c,i = SieveOfEratosthenes(100000000)
print("Total prime numbers in range:", c )
print("largest prime", i)
t1 = time.time()
print("Time required:", t1 - t0)
```

Output:

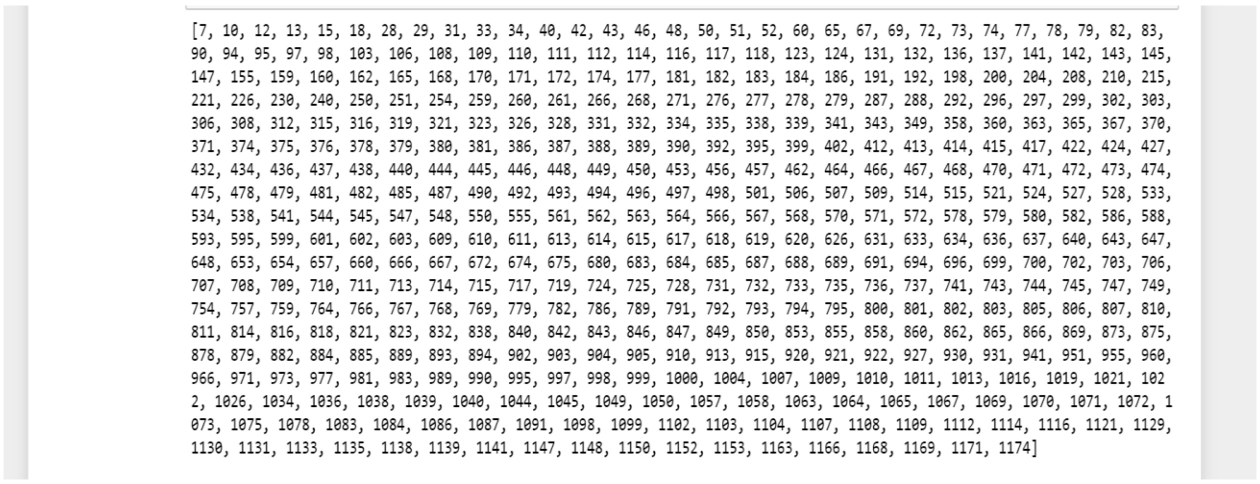


```
Total prime numbers in range: 5761455
largest prime 99999271
Time required: 79.67130708694458
```

2. Code for finding Primitive root (min 10 digits)

```
from math import gcd as bltin_gcd

def primRoots(modulo):

    required_set = {num for num in range(1, modulo) if bltin_gcd(num, modulo) }

    return [g for g in range(1, modulo) if required_set == {pow(g, powers, modulo)

            for powers in range(1, modulo)}]

print(primRoots(1181))
```

Output:

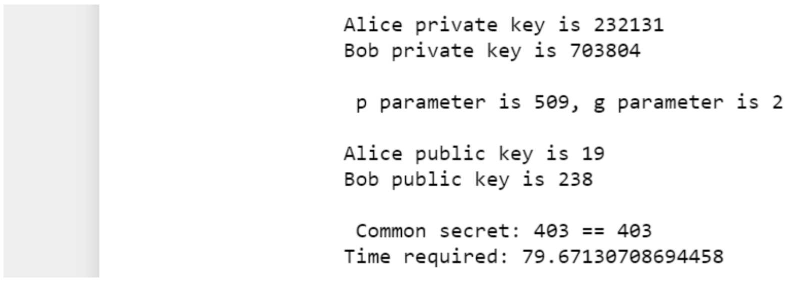


```
[7, 10, 12, 13, 15, 18, 28, 29, 31, 33, 34, 40, 42, 43, 46, 48, 50, 51, 52, 60, 65, 67, 69, 72, 73, 74, 77, 78, 79, 82, 83,
90, 94, 95, 97, 98, 103, 106, 108, 109, 110, 111, 112, 114, 116, 117, 118, 123, 124, 131, 132, 136, 137, 141, 142, 143, 145,
147, 155, 159, 160, 162, 165, 168, 170, 171, 172, 174, 177, 181, 182, 183, 184, 186, 191, 192, 198, 200, 204, 208, 210, 215,
221, 226, 230, 240, 250, 251, 254, 259, 260, 261, 266, 268, 271, 276, 277, 278, 279, 287, 288, 292, 296, 297, 299, 302, 303,
306, 308, 312, 315, 316, 319, 321, 323, 326, 328, 331, 332, 334, 335, 338, 339, 341, 343, 349, 358, 360, 363, 365, 367, 370,
371, 374, 375, 376, 378, 379, 380, 381, 386, 387, 388, 389, 390, 392, 395, 399, 402, 412, 413, 414, 415, 417, 422, 424, 427,
432, 434, 436, 437, 438, 440, 444, 445, 446, 448, 449, 450, 453, 456, 457, 462, 464, 466, 467, 468, 470, 471, 472, 473, 474,
475, 478, 479, 481, 482, 485, 487, 490, 492, 493, 494, 496, 497, 498, 501, 506, 507, 509, 514, 515, 521, 524, 527, 528, 533,
534, 538, 541, 544, 545, 547, 548, 550, 555, 561, 562, 563, 564, 566, 567, 568, 570, 571, 572, 578, 579, 580, 582, 586, 588,
593, 595, 599, 601, 602, 603, 609, 610, 611, 613, 614, 615, 617, 618, 619, 620, 626, 631, 633, 634, 636, 637, 640, 643, 647,
648, 653, 654, 657, 660, 666, 667, 672, 674, 675, 680, 683, 684, 685, 687, 688, 689, 691, 694, 696, 699, 700, 702, 703, 706,
707, 708, 709, 710, 711, 713, 714, 715, 717, 719, 724, 725, 728, 731, 732, 733, 735, 736, 737, 741, 743, 744, 745, 747, 749,
754, 757, 759, 764, 766, 767, 768, 769, 779, 782, 786, 789, 791, 792, 793, 794, 795, 800, 801, 802, 803, 805, 806, 807, 810,
811, 814, 816, 818, 821, 823, 832, 838, 840, 842, 843, 846, 847, 849, 850, 853, 855, 858, 860, 862, 865, 866, 869, 873, 875,
878, 879, 882, 884, 885, 889, 893, 894, 902, 903, 904, 905, 910, 913, 915, 920, 921, 922, 927, 930, 931, 941, 951, 955, 960,
966, 971, 973, 977, 981, 983, 989, 990, 995, 997, 998, 999, 1000, 1004, 1007, 1009, 1010, 1011, 1013, 1016, 1019, 1021, 102
2, 1026, 1034, 1036, 1038, 1039, 1040, 1044, 1045, 1049, 1050, 1057, 1058, 1063, 1064, 1065, 1067, 1069, 1070, 1071, 1072, 1
073, 1075, 1078, 1083, 1084, 1086, 1087, 1091, 1098, 1099, 1102, 1103, 1104, 1107, 1108, 1109, 1112, 1114, 1116, 1121, 1129,
1130, 1131, 1133, 1135, 1138, 1139, 1141, 1147, 1148, 1150, 1152, 1153, 1163, 1166, 1168, 1169, 1171, 1174]
```

3. Code for DH (Taking random private keys)

```
from random import getrandbits
from random import randint
import sys

def is_prime_calc(num):
    return all(num % i for i in range(2, num))

def is_prime(num):
    return is_prime_calc(num)

def get_random_prime():
    while True:
        n = getrandbits(12) + 3;
        if is_prime(n):
            return n

def gcd(a,b):
    while a != b:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a

def primitive_root(modulo):
    required_set = set(num for num in range (1, modulo) if gcd(num, modulo) == 1)
    for g in range(1, modulo):
        actual_set = set(pow(g, powers) % modulo for powers in range (1, modulo))
        if required_set == actual_set:
            return g

# Generating private keys
alice_private = randint(999, 999999)
print ('Alice private key is %d' % alice_private)
bob_private = randint(999, 999999)
print ('Bob private key is %d' % bob_private)

# Generating p-g parameters
p = get_random_prime()
g = primitive_root(p)

print ('\n p parameter is %d, g parameter is %d \n' % (p, g))

# Generating public keys
alice_public = pow(g, alice_private) % p
bob_public = pow(g, bob_private) % p

print ('Alice public key is %d' % alice_public)
print ('Bob public key is %d' % bob_public)

alice_key = (pow(bob_public, alice_private)) % p
bob_key = (pow(alice_public, bob_private)) % p

print ('\n Common secret: %d == %d' % (alice_key, bob_key))
print("Time required:", t1 - t0)
```

Output:

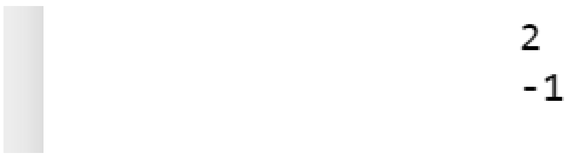


```
Alice private key is 232131
Bob private key is 703804

 p parameter is 509, g parameter is 2

Alice public key is 19
Bob public key is 238

 Common secret: 403 == 403
Time required: 79.67130708694458
```

4. Code to Find an integer **k** such that $a^k \equiv b \pmod{m}$ where a and m are relatively prime.

```
import math;

def discreteLogarithm(a, b, m):

    n = int(math.sqrt (m) + 1);

    # Calculate a ^ n
    an = 1;
    for i in range(n):
        an = (an * a) % m;

    value = [0] * m;

    # Store all values of a^(n*i) of LHS
    cur = an;
    for i in range(1, n + 1):
        if (value[ cur ] == 0):
            value[ cur ] = i;
        cur = (cur * an) % m;

    cur = b;
    for i in range(n + 1):

        # Calculate (a ^ j) * b and check
        # for collision
        if (value[cur] > 0):
            ans = value[cur] * n - i;
            if (ans < m):
                return ans;
        cur = (cur * a) % m;

    return -1;

# Driver code
a = 200;
b = 3;
m = 5;
print(discreteLogarithm(a, b, m));

a = 350;
b = 7;
m = 11;
print(discreteLogarithm(a, b, m));
```

Output:



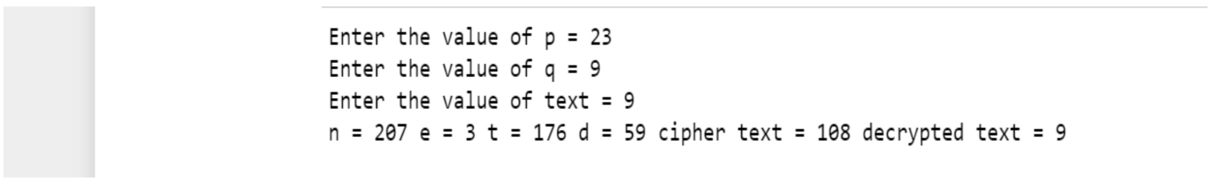
```
2
-1
```

5. Code

```
from decimal import Decimal


def gcd(a,b):
    if b==0:
        return a
    else:
        return gcd(b,a%b)
p = int(input('Enter the value of p = '))
q = int(input('Enter the value of q = '))
no = int(input('Enter the value of text = '))
n = p*q
```

```
t = (p-1)*(q-1)


for e in range(2,t):
    if gcd(e,t)== 1:
        break



for i in range(1,10):
    x = 1 + i*t
    if x % e == 0:
        d = int(x/e)
        break
ctt = Decimal(0)
ctt =pow(no,e)
ct = ctt % n


dtt = Decimal(0)
dtt = pow(ct,d)
dt = dtt % n


print('n = '+str(n)+' e = '+str(e)+' t = '+str(t)+' d = '+str(d)+' cipher text = '+str(ct)+' decrypted text = '+str(dt))
```

Output:

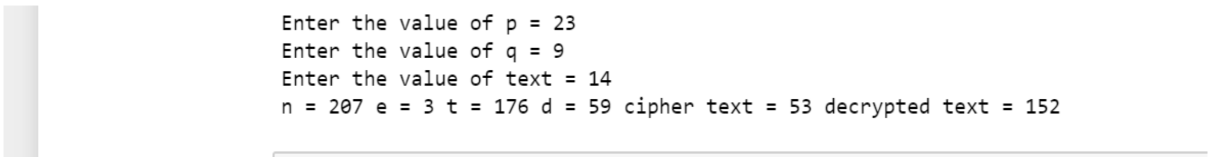


```
Enter the value of p = 23
Enter the value of q = 9
Enter the value of text = 9
n = 207 e = 3 t = 176 d = 59 cipher text = 108 decrypted text = 9
```

```
from decimal import Decimal

def gcd(a,b):
    if b==0:
        return a
    else:
        return gcd(b,a%b)
p = int(input('Enter the value of p = '))
q = int(input('Enter the value of q = '))
no = int(input('Enter the value of text = '))
n = p*q
t = (p-1)*(q-1)

for e in range(2,t):
    if gcd(e,t)== 1:
        break



for i in range(1,10):
    x = 1 + i*t
    if x % e == 0:
        d = int(x/e)
        break
ctt = Decimal(0)
ctt =pow(no,e)
ct = ctt % n

dtt = Decimal(0)
dtt = pow(ct,d)
dt = dtt % n
```

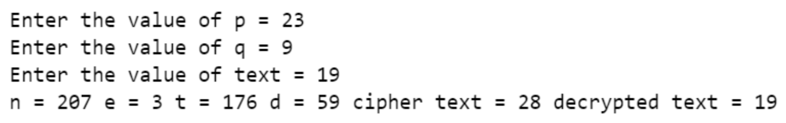
print('n = '+str(n)+' e = '+str(e)+' t = '+str(t)+' d = '+str(d)+' cipher text = '+str(ct)+' decrypted text = '+str(dt))

Output:


```
Enter the value of p = 23
Enter the value of q = 9
Enter the value of text = 14
n = 207 e = 3 t = 176 d = 59 cipher text = 53 decrypted text = 152
```

```
from decimal import Decimal

def gcd(a,b):
    if b==0:
        return a
    else:
        return gcd(b,a%b)
p = int(input('Enter the value of p = '))
q = int(input('Enter the value of q = '))
no = int(input('Enter the value of text = '))
n = p*q
t = (p-1)*(q-1)

for e in range(2,t):
    if gcd(e,t)== 1:
        break


for i in range(1,10):
    x = 1 + i*t
    if x % e == 0:
        d = int(x/e)
        break
ctt = Decimal(0)
ctt =pow(no,e)
ct = ctt % n

dtt = Decimal(0)
dtt = pow(ct,d)
dt = dtt % n

print('n = '+str(n)+' e = '+str(e)+' t = '+str(t)+' d = '+str(d)+' cipher text = '+str(ct)+' decrypted text = '+str(dt))
```

Output:



```
Enter the value of p = 23
Enter the value of q = 9
Enter the value of text = 19
n = 207 e = 3 t = 176 d = 59 cipher text = 28 decrypted text = 19
```