2.3 Public Key Cryptography

This project requires an understanding of the Part IA course Numbers and Sets.

1 The RSA Algorithm

The essence of Public Key Cryptography is to make it easy for people to send securely encrypted messages to one another without the need for each pair to agree beforehand on a secret code. Encryption and decryption is done via a publicly known algorithm which makes use of a *key*. Each person privately constructs two keys K_1 and K_2 which are, in effect, inverses of each other; that is to say, a message encrypted with K_1 can be decrypted using K_2. The person then publishes K_1 to the whole community — it becomes that person's *public key*. The key K_2 is kept (very) private. Then, to send a message to any given person, the message is encrypted using their public key. Only that person can decrypt the message because only that person has the decryption key K_2.

In order for a public key algorithm to be effective, it must be feasible for a person to construct two inverse keys K_1 and K_2, but it must be infeasible for anyone to work out the value of K_2 given only the value of K_1.

In the RSA system (so called after its inventors, Rivest, Shamir and Adleman) an individual follows this recipe. Choose two large primes p and q and compute $n = pq$. Then choose a number $e < n$ which is coprime to $\varphi(n) = (p-1)(q-1)$. Then find a number d such that $ed \equiv 1 \pmod{\varphi(n)}$. The public key is the pair (n, e) and the private key is (n, d). A message m sent to this person would be encrypted as $c = m^e \pmod{n}$. The individual for whom the message was intended would decrypt c as $m = c^d \pmod{n}$.

The RSA algorithm works because it is feasible for a user to find a couple of 100-digit numbers which are almost certainly prime, but it is thought to be impossible to factorize a 200-digit number in a reasonable time. Factorization of n would enable an enemy to find d from e in just the same way that the user did. There are other possible attacks but they appear to be (though are not proved to be) as difficult as factorization.

2 Working with integers

The quantities of interest in this project are natural numbers. MATLAB, unfortunately, is designed for those who think that real numbers are the only really natural numbers. This has an effect on how easy it is to get MATLAB to do integer arithmetic and on how large the numbers can be before errors start to occur.

Computers typically store a number as a string of 32 or 64 binary digits (bits), which can, if desired, represent integers up to 2^{32} or 2^{64}. If the computer understands that we are working only with integers, then the representation is very efficient, and moreover, if it is asked to compute, say, 14/3 it will respond with 4, this being the correct integer value. In other words, the machine will perform modular arithmetic simply and with complete accuracy, which is ideal.

MATLAB, though, with its incorrigible real-world view, uses the 64 bits to store what it believes is a real number. This means both that the space available for the integer part is limited, and that rounding errors will occur. If you ask for the value of 14/3 it will produce a reasonable but incorrect result. You can force MATLAB to recognize integers via the `int32` or `int64` commands (see the CATAM manual). However the results are not always what you might expect, and for

this project it is suggested that you go with the flow and just allow MATLAB to think it is working with reals.

The only practical consequence of the above discussion that needs to be borne in mind is that MATLAB can handle integers in this way only up to about 15 decimal digits. (Try `eps(10^15)` and `eps(10^16)`. Try also `10^8 * 10^8 - (10^16-1)`.) So you must avoid calculations involving integers exceeding 10^{15}. This is a pity, because it is easy to crack RSA with numbers of this size. The adventurous might like to find out how to handle larger numbers in MATLAB; trying RSA on 50-digit numbers brings home much more vividly what is going on. But this remark is for just for interest, and you will get no extra credit for pursuing it.

You will find it helpful to type `format longg` at the beginning of a MATLAB session: this encourages MATLAB to print integers larger than 10^9 properly, rather than in engineering format.

3 Prime Numbers and Factorization

The easiest way to test a small number n for primality is by *trial division*: divide by every number up to n. The MATLAB function `mod` is useful for this: `mod(a,b)` gives the value of a modulo b. The builtin functions `isprime()`, `factor()` and `primes()` should not be used in the next two questions except, if desired, for checking answers.

> **Question 1** This method takes a while for a 10-digit number; what simple modification will speed it up? Write a function to apply this method (and exhibit sample output).

In subsequent questions, an "arithmetic operation" can be taken to be the addition, subtraction, multiplication or division of two integers.

> **Question 2** Modify your improved algorithm from Question 1 so as to find the complete prime factorization of a number n. Try your algorithm on a few examples.
>
> Your algorithm may be much quicker for some types of numbers than for others of similar size. Estimate the complexity of your algorithm — that is, estimate the number of arithmetic operations needed in the worst case, as a function of n. (You may wish to consider first some extreme cases.) Can you prove that your estimate is correct?

4 Linear Congruences

One way to find the highest common factor of two numbers is to factorize them both and then form the product of the common prime factors.

Euclid's Algorithm is a more efficient way to compute the highest common factor of two numbers. Importantly, it also provides a means to find integers u and v such that $au + bv = \mathrm{hcf}(a, b)$.

> **Question 3** Implement Euclid's Algorithm. Thereby find the highest common factor of each of the following pairs of numbers, and express it as a linear combination of the original two numbers.
>
> 1 996 245 783 and 192 784 863 2 825 746 811 and 758 295 345
> 249 508 543 104 and 338 063 357 376 249 508 543 140 and 338 063 357 367

Question 4 Describe clearly how Euclid's algorithm can be used to find all of the solutions in the unknown x to the linear congruence $ax \equiv b \pmod{m}$.

Question 5 Implement a routine to solve the linear congruence $ax \equiv b \pmod{m}$.

Find all solutions to each of the following congruences. If none exist, state why not.

$$\begin{aligned} 146295x &\equiv 2017 \pmod{313567} \\ 93174x &\equiv 2015 \pmod{267975} \\ 113314x &\equiv 2014 \pmod{660115} \end{aligned}$$

5 Finding Inverses and Breaking RSA

Let $n = pq$ where p and q are primes. Then $\varphi(n) = (p-1)(q-1)$ where $\varphi(n)$ is Euler's function. If e is coprime to $\varphi(n)$ then there exists a d such that $ed \equiv 1 \pmod{\varphi(n)}$. The program you developed for Question 5 should enable you to find such a d for a given e.

Question 6 Given n and e, but not p or q, approximately how many arithmetic operations does your program need to find p and q? (Remember to justify your answers.)

Given p and q, and hence $\varphi(n)$, approximately how many operations are needed to find d?

Question 7 Write a program to compute the private decryption key from a given public encryption key. Find the decryption keys corresponding to the following:

$$\begin{array}{ccc} (1\,764\,053\,131, 103\,471) & (1\,805\,760\,301, 39\,871\,477) & (9\,976\,901\,028\,181, 837\,856\,358\,917) \\ (1\,723\,466\,867, 692\,581\,937) & (6\,734\,071\,952\,813, 2017) & (1\,603\,982\,333, 927\,145) \end{array}$$

6 Decrypting RSA messages

Question 8 Write a program to convert an encrypted number $c = m^e \pmod{n}$ into the original $m = c^d \pmod{n}$, where $0 \leqslant m < n$ is some integer.

State, with justification, the greatest number of digits that n can have for which your program can be trusted to work.

My public key is $(937513, 638471)$. Our community has agreed to use the encoding $00 \leftrightarrow$ `space`, $01 \leftrightarrow$ `a`, $\ldots$, $26 \leftrightarrow$ `z`, $27 \leftrightarrow$., $28 \leftrightarrow$:, $29 \leftrightarrow$ ′, and to encrypt blocks of 3 letters at a time (so that, for example, the message "`have a nice day`" is encoded as the five numbers 080122 050001 001409 030500 040125 before encryption).

Question 9 I receive the encrypted message

179232	006825	263565	126615	474921	750809	900050	009287
554344	413204	757176	066356	716784	382286	696566	610518
510930	459403	922484	390971	773831	655925	633419	519880

What is the message?

[You do not need to write programs to convert a string of characters into the appropriate sequence of integers or vice versa.]

2.3 Public Key Cryptography

Question 1

To reduce the number of calculations in the *trial division test* (divide by every number up to n) we make the following adaptation: divide n by every number up to $\lfloor\sqrt{n}\rfloor$ (where $\lfloor x\rfloor$ is the floor function of x).
Why does this adapted test work? Suppose n has a factors a and b such that $n = ab$ and $a \leq b$, then we must have $a \leq \sqrt{n} \leq b$. So if n has no factors less then or equal to $\lfloor\sqrt{n}\rfloor$ then n has no factors.
(We need only test up to $\lfloor\sqrt{n}\rfloor$ as either $\lfloor\sqrt{n}\rfloor = \sqrt{n}$ or there will be no integers in the interval $(\lfloor\sqrt{n}\rfloor, \sqrt{n}.)$

I have designed a program using the *adapted trial division test* (called `Question_1` in project folder - see page 7 for script). In my program I generate a table showing the result of each calculation to identify if there are any errors in the code. Here are some sample outputs of the program.

```
Input number n = 2
the number is prime

Input number n = 4
k  mod(n,k)
2     0

the number is NOT prime

Input number n = 5
k  mod(n,k)
2     1

the number is prime

Input number n = 11
k  mod(n,k)
2     1
3     2

the number is prime

Input number n = 121
k  mod(n,k)
2     1
3     1
4     1
5     1
6     1
7     2
8     1
9     4
10     1
11     0

the number is NOT prime
```

```
Input number n = 983
k  mod(n,k)
2     1
3     2
4     3
5     3
6     5
7     3
8     7
9     2
10     3
11     4
12    11
13     8
14     3
15     8
16     7
17    14
18    11
19    14
20     3
21    17
22    15
23    17
24    23
25     8
26    21
27    11
28     3
29    26
30    23
31    22

the number is prime
```

```
Input number n = 10697
k  mod(n,k)
2     1
3     2
4     1
5     2
6     5
7     1
8     1
9     5
10     7
11     5
12     5
13    11
14     1
15     2
16     9
17     4
18     5
19     0

the number is NOT prime

Input number n = 12340068
k  mod(n,k)
2     0

the number is NOT prime

Input number n = 123456789
k  mod(n,k)
2     1
3     0

the number is NOT prime
```

Question 2

Before I proceed I must prove a statement.

Claim: Suppose n is not prime. Then the algorithm `Question_1` terminates when k is the smallest factor of n and is prime.

Proof: Suppose the algorithm terminates at $k = k^*$ (where k is the number we apply mod (n, k) to in the algorithm.
If there were a smaller factor of n then the alogrithm would terminate before $k = k^*$.
So k^* must be the smallest factor of n.
Assume that k^* is not prime. Then $k^* = ab$ with $2 \leq a \leq b$.
But then a divides n so the algorithm would terminate at a.
So the assumption k^* is not prime contradicts the algorithm terminates at $k = k^*$
So k^* is prime. □

Using this fact I can now write an algorithm which determines if n is prime or not in exactly the same way as in `Question_1`. If n is prime the algorithm terminates. If not we store the first value k such that $n \equiv 0(\mod k)$ and calculate $\frac{n}{k}$. Then we repeat the algorithm, this time to determine if $\frac{n}{k}$ is prime, etc. With this in mind we produce an program to find the prime factors of a number (called `Question_2` in project folder - see page 8 for script). The program outputs the number of arithmetic operations to aid me in the next part of this question. Below are some examples of the program at work:

```
Input number n=11
The prime factors of n are:
11

Arithmetic Operations=7
```

```
Input number n=100
The prime factors of n are:
2
2
5
5

Arithmetic Operations=16
```

```
Input number n=121
The prime factors of n are:
11
11

Arithmetic Operations=30
```

```
Input number n=256
The prime factors of n are:
2
2
2
2
2
2
2
2

Arithmetic Operations=15
```

```
Input number n=200006592457
The prime factors of n are:
7
7
7
583109599

Arithmetic Operations=72445
```

```
Input number n=200132629100
The prime factors of n are:
2
2
5
5
2001326291

Arithmetic Operations=134214
```

```
Input number n=200506629418
The prime factors of n are:
2
100253314709

Arithmetic Operations=949884
```

```
Input number n=200514703783
The prime factors of n are:
149
167
181
211
211

Arithmetic Operations=636
```

```
Input number n=200560490130
The prime factors of n are:
2
3
5
7
11
13
17
19
23
29
31

Arithmetic Operations=102
```

```
Input number n=208999145189
The prime factors of n are:
5479
5563
6857

Arithmetic Operations=16688
```

```
Input number n=209897547617
The prime factors of n are:
 209897547617

Arithmetic Operations=1374433
```

```
Input number n=214996924823
The prime factors of n are:
433723
495701

Arithmetic Operations=1301166
```

The examples on the previous page suggest that in general numbers which are the product of lots of small prime numbers have a low number of arithmetic operations whereas numbers which are the product of very few prime numbers (or numbers which are prime) require a large number of arithmetic operations. I will estimate the complexity as $O(\sqrt{n})$. I now aim to prove this is the case.

To calculate the number of arithmetic operations we must be clear exactly how the algorithm works. Hence I produce a step by step procedure which the algorithm `Question_2` conducts:

1. Set $k = 2$.
2. Calculate k^2.
3. If $k^2 > n$ record n as the final prime factor and terminate the algorithm.
4. Calculate $n \pmod k$. Then:
   (a) if $n \equiv 0 \pmod k$ then replace n with n/k. Save this value of k as one of the prime factors and then return to Step 3.
   (b) else replace k with $k + 1$ and return to Step 2.

Note: We require 1 arithmetic operation in steps 2, 3 and 4

. Consider $n = p_1 p_2 ... p_k$ with $p_1 \leq p_2 \leq ... \leq p_k$.
The number of arithmetic operations needed to calculate p_1 is $3(p_1 - 1)$ as we must conduct steps 2, 3 and 4 for $k = 2, 3, ..., p_1$.
Then we require $3(p_2 - p_1 + 1) - 1$ operations to calculate p_2 as we conduct steps 2, 3 and 4 for $k = p_1 + 1, ...p_2$ and in addition conduct steps 3 and 4 for $k = p_1$.
Likewise to find p_i for $i < k$ we require $3(p_i - p_{i-1} + 1) - 1$ arithmetic operations.
There are two ways the algorithm will find the final prime number and terminate. The first is that $p_{k-1}^2 > n$ as so not more arithmetic operations are required. The second is that $p_{k-1}^2 < n$ and so we need $3(\lfloor\sqrt{p_k}\rfloor - p_{k-1}) + O(1)$.

So the number of arithmetic operations required is:

$$\begin{aligned} 3(\max(p_{k-1}, \lfloor\sqrt{p_k}\rfloor)) + O(k) + O(1) &= O(\max(p_{k-1}, \lfloor\sqrt{p_k}\rfloor)) + O(k) \\ &\leq O(\max(p_{k-1}, \lfloor\sqrt{p_k}\rfloor)) + O(\log_2 k) \\ &\leq O(\max(p_{k-1}, \lfloor\sqrt{p_k}\rfloor)) + O(\sqrt{n}) \\ &\leq O(\sqrt{n}) + O(\sqrt{n}) \\ &= O(\sqrt{n}) \end{aligned}$$

This upper bound is attained when n is prime. So the complexity is $O(\sqrt{n})$.

Question 3

I implement Euclid's Algorithm in `Question_3` which is found on page 9. The algorithm outputs the hcf of the two numbers, both in the form $au + bv$ and just a number.

(i)
```
Input a = 1996245783
Input b = 192784863
hcf = 1996245783*(-11108123) + 192784863*(115022224) = 3
```

(ii)
```
Input a = 2825746811
Input b = 758295345
hcf = 2825746811*(-28353319) + 758295345*(105657118) = 1
```

(iii)
```
Input a = 249508543104
Input b = 338063357376
hcf = 338063357376*(-356165) + 249508543104*(482574) = 46656
```

(iv)
```
Input a = 249508543140
Input b = 338063357367
hcf = 338063357367*(-8485693393) + 249508543140*(11497409916) = 9
```

Question 4

Before I start explaining how to solve $ax \equiv b \pmod{m}$ I will prove something. We will use this statement as a test to determine if an equation can be solved.

Claim: There is a solution to $ax \equiv b (\text{mod } m)$ if and only if $\text{hcf}(a,m)|b$. If $d = \text{hcf}(a,m)|b$ then the solution is the unique solution to $\frac{a}{d}x \equiv \frac{b}{d}(\text{mod } \frac{m}{d})$.

Proof: Let $d = \text{hcf}(a,m)$.
If there is a solution to $a \equiv b \pmod{m}$ then $m|ax - b$.
So $d|ax - b$ and $d|b$.
Now suppose $d|b$ and $ax \equiv b \pmod{m}$.
$\Leftrightarrow ax - b = km$ for some $k \in \mathbb{Z}$.
$\Leftrightarrow d\frac{a}{d}x - d\frac{b}{d} = dk\frac{m}{d}$ for some $k \in \mathbb{Z}$ where we observe each fraction is an integer.
$\Leftrightarrow \frac{a}{d}x - \frac{b}{d} = k\frac{m}{d}$ for some $k \in \mathbb{Z}$.
$\Leftrightarrow \frac{a}{d}x \equiv \frac{b}{d} \pmod{\frac{m}{d}}$.

□

Below are the steps one follows in order to solve $ax \equiv b \pmod{m}$:

1. Use Euclid's Algorithm to find $\text{hcf}(a,m)$. Express the highest common factor as a linear combination of a and m, i.e. $au + mv = hcf(a,m)$. Therefore $\frac{a}{\text{hcf}(a,m)}u + \frac{m}{\text{hcf}(a,m)}v = 1$.

2. Check if $\text{hcf}(a,m)|b$. If this does not hold there are no solutions to the linear congruence by the statement proven above. If $\text{hcf}(a,m)|b$ is true proceed to step 3.

3. Observe $\frac{a}{\text{hcf}(a,m)}u \equiv 1 \left(\text{mod } \frac{m}{\text{hcf}(a,m)}\right)$.
   Therefore $\frac{a}{\text{hcf}(a,m)}ux \equiv \frac{b}{\text{hcf}(a,m)}u \left(\text{mod } \frac{m}{\text{hcf}(a,m)}\right)$.
   So $x \equiv \frac{b}{\text{hcf}(a,m)}u \left(\text{mod } \frac{m}{\text{hcf}(a,m)}\right)$ is the set of solutions to the linear congruence.

Question 5

Code found on page 10 labelled `Question_5` in the project folder.

(i)
```
Input a = 146295
Input b = 2017
Input m = 313567
hcf = 313567*(64273) + 146295*(-137762) = 1
x = 267975 (mod 313567)
```

(ii)
```
Input a = 93174
Input b = 2015
Input m = 267975
hcf = 267975*(-9691) + 93174*(27872) = 3
This problem has no solutions because hcf(a,m) does not divide b
```

(iii)
```
Input a = 113314
Input b = 2014
Input m = 660115
hcf = 660115*(791) + 113314*(-4608) = 53
x = 11721 (mod 12455)
```

Note: In part (ii) since $\text{hcf}(a,m)$ does not divide b we can appeal to the claim from Question 4 and conclude there are no solutions to $ax \equiv b \pmod{m}$.

Question 6

To factorize $n = pq$ with $p \leq q$ we follow the following process:

1. Set $k_1 = 2$.
2. Calculate $n \pmod{k_i}$ (with i initially 1). Then:
   - if $n \equiv 0 \pmod{k_i}$ then $p = k_i$ and $q = n/p$.
   - else return to step 2 with $k_{i+1} = k_i + 1$.

Each iteration requires two arithmetic operations, the first to calculate $n \pmod{p_i}$ and the second to calculate either n/p or $p_i + 1$. There are $p - 1$ iterations of this process. Therefore there are $2(p-1) = O(p)$ arithmetic operations.

Now I will list the process required to calculate d (note this is essentially Euclid's Algorithm):

1. Let $m := \phi(n)$, $b := e$, $B_{-1} = 0 = M_0$ and $M_{-1} = 1 = B_0$.
2. Set $r \equiv m \pmod{b}$.
3. Set $q = \dfrac{b-r}{m}$.
4. Let $B_j := qB_{j-1} + B_{j-2}$ and $M_j := qM_{j-1} + M_{j-2}$.
5. If $r \equiv 0 \pmod{\phi(n)}$ then proceed to Step 6. Else redefine $b := m$ and $m := r$ and return to Step 2.
6. Suppose there are k steps to the algorithm.
   Then $eM_{k-1} - \phi(n)B_{k-1} = (-1)^{k-1}$.
   Therefore $eM_{k-1} - \phi(n)E_{k-1} \equiv eM_{k-1} \equiv \pm 1 \pmod{\phi(n)}$.
   So $d \equiv \pm M_{k-1} \pmod{\phi(n)}$.

So the algorithm is $O(k)$ where k is the number of steps. I will look for an upper bound on k.

First observe $e < \phi(n)$. It follows that $r < e$.
Then by Euclid's Algorithm $\phi(n) = qe + r$ where $q \leq 1$.
Therefore $2r < e + r \leq \phi(n)$ or $2r \leq \phi(n)$.
Hence $re \leq \dfrac{e\phi(n)}{2}$.
So each stage reduces the product of mb by a half. After k stages the product of mb is non-zero so $2^k \leq ab$.
Hence $k = \log_2 a + \log_2 b$.

So the algorithm has complexity $O(\log_2 a) + O(\log_2 b)$.

Question 7

Code found on page 12 labelled `Question_7` in the project folder.

(i)
```
Input number n = 1764053131
Input number e = 103471
p = 40013
q = 44087
hcf(e, phi) = 1763969032*(-11238) + 103471*(191584927) = 1
d = 191584927 (mod 1764053131)
```

(ii)
```
Input number n = 1805760301
Input number e = 39871477
p = 39019
q = 46279
hcf(e, phi) = 1805675004*(7791960) + 39871477*(-352877507) = 1
d = 1452797497 (mod 1805760301)
```

(iii)
```
Input number n = 9976901028181
Input number e = 837856358917
p = 1234627
q = 8080903
hcf(e, phi) = 9976891712652*(-324559277272) + 837856358917*(3864734962285) = 1
d = 3864734962285 (mod 9976901028181)
```

(iv)
```
Input number n = 1723466867
Input number e = 692581937
p = 37831
q = 45557
hcf(e, phi) = 1723383480*(-90521525) + 692581937*(225248873) = 1
d = 225248873 (mod 1723466867)
```

(v)
```
Input number n = 6734071952813
Input number e = 2017
p = 2020223
q = 3333331
hcf(e, phi) = 6734066599260*(797) + 2017*(-2660907823307) = 1
d = 4073158775953 (mod 6734071952813)
```

(vi)
```
Input number n = 1603982333
Input number e = 927145
p = 20599
q = 77867
hcf(e, phi) = 1603883868*(49102) + 927145*(-84942383) = 1
d = 1518941485 (mod 1603982333)
```

Question 8

The program I used to convert $c = m^e \pmod{n}$ into $m = e^d \pmod{n}$ is `Question_8` (found on page 14). The program first finds the decryption key d. Then it calculates m_d where $m_i \equiv cm_{i-1} \pmod{m}$ for $i > 1$ and $m_1 = 1$. This number m_d is our message.

As stated in the project manual, we should avoid using integers exceeding 10^{15}. Therefore we cannot allow the program to multiply two which both exceed $10^{7.5}$. Since $0 \leq c, m_i < n$ we can prevent this from happening by upper bounding n by $10^{7.5} \approx 31622776.6$. Since $n \in \mathbb{Z}^+$ we can upper bound n by 31622776. It follows that we can always trust the program to work when n has at most 7 digits.

Question 9

I use `Question_8` to decrypt the message with `n = 937513` and `e = 638471`. I input each value of `c` listed in the project manual. We may use this program because n is only 6 digits. The decrypted message is below:

092913	001415	200018	050112	122500	091400	010003	080505
i ' m	_ n o	t _ r	e a l	l y _	i n _	a _ c	h e e
190500	131515	042700	141523	002008	012029	190023	080120
s e _	m o o	d . _	n o w	_ t h	a t '	s _ w	h a t
000900	030112	120001	140005	070700	190114	042309	030827
_ i _	c a l	l _ a	n _ e	g g _	s a n	d w i	c h .

Putting this all together (and adding capital letters where appropriate places):
I'm not really in a cheese mood. Now that's what I call an egg sandwich.

Appendix: Question 1

This is labelled `Question_1` in folders and used to determine if n is prime.

```
%clear; clc;
format longg

n = input('Input number n = ');
k =2;
j=0;
logic = 0;
table = [];

while k^2 <= n && logic == 0;
j=1;
m = mod(n,k);
table = [table; k m];

if m == 0;
logic = 1;
else
k = k+1;
end

end
if j==1;
disp('     k  mod(n,k)')
disp(table);
end

if logic == 0;
display('the number is prime')
else
display('the number is NOT prime')
end
```

Appendix: Question 2

This is labelled `Question_2` in folders and used to determine if n is prime.

```
clear; clc;
format longg

n = input('Input number n=');
k =2;
k_2 = k^2;
Arithmetic_Operations = 1; %initial k^2 with k = 2
table = [];
number_of_steps=0;

while k_2 <= n;
number_of_steps=number_of_steps+1;
m = mod(n,k);
Arithmetic_Operations = Arithmetic_Operations +1; %mod(n,k)

if m == 0;
table = [table; k];
n = n/k;
Arithmetic_Operations = Arithmetic_Operations +1; %n/k
else
k = k+1;
Arithmetic_Operations = Arithmetic_Operations +1; %k=k+1
k_2 = k^2;
Arithmetic_Operations = Arithmetic_Operations +1; %k^2
end
end

table = [table; n];

disp('The prime factors of n are:')
disp(table);

X = sprintf('Number of Steps = %i',number_of_steps);
%disp(X);

Y = sprintf('Arithmetic Operations=%i',Arithmetic_Operations);
disp(Y);
```

Appendix: Question 3

```
%clear; clc;
format longg

a_input = input('Input a = ');
b_input = input('Input b = ');
logic = 0;
A_i = 0;
B_i = 1;
A_j = 1;
B_j = 0;
counter = 0;

if abs(a_input)>abs(b_input);
a = a_input;
a_0 = a_input;
b = b_input;
b_0 = b_input;
else
a = b_input;
a_0 = b_input;
b = a_input;
b_0 = a_input;
end

while logic ==0;
counter = counter + 1;
r = mod(a,b);
q = (a-r)/b;
A_k = q*A_j + A_i;
B_k = q*B_j + B_i;
if r ==0;
logic = 1;
else
A_i = A_j;
B_i = B_j;
A_j = A_k;
B_j = B_k;
a = b;
b = r;
end
end

sign = (-1)^counter;

if sign == 1;
u = B_j;
v = -A_j;
else
u = -B_j;
v = A_j;
end

X = sprintf('hcf = %i*(%i) + %i*(%i) = %i',a_0,u,b_0,v,b);
disp(X);
```

Appendix: Question 5

```
clear; clc;
format longg

a_input = input('Input a = ');
b_input = input('Input b = ');
m_input = input('Input m = ');
logic = 0;
A_i = 0;
M_i = 1;
A_j = 1;
M_j = 0;
counter = 0;

if abs(a_input)>abs(m_input);
a = a_input;
a_0 = a_input;
m = m_input;
m_0 = m_input;
else
a = m_input;
a_0 = m_input;
m = a_input;
m_0 = a_input;
end

while logic ==0;
counter = counter + 1;
r = mod(a,m);
q = (a-r)/m;
A_k = q*A_j + A_i;
M_k = q*M_j + M_i;
if r ==0;
logic = 1;
else
A_i = A_j;
M_i = M_j;
A_j = A_k;
M_j = M_k;
a = m;
m = r;
end
end

sign = (-1)^counter;

if sign == 1;
u = M_j;
v = -A_j;
else
u = -M_j;
v = A_j;
end

X = sprintf('hcf = %i*(%i) + %i*(%i) = %i',a_0,u,m_0,v,m);
disp(X);
```

```
d = m;
b = b_input;

divide = mod(b,d);

if b < d;
display('This problem has no solutions because b/d < 1')
elseif divide > 0;
display('This problem has no solutions because hcf(a,m) does not divide b')
else
b_prime = b_input/d;
m_prime = m_input/d;
if abs(a_input)>abs(m_input);
x = mod(b_prime*u, m_prime);
else
x = mod(b_prime*v, m_prime);
end
Y = sprintf('x = %i (mod %i)',x,m_prime);
disp(Y);
end
```

Appendix: Question 7

```
clear; clc;
format longg

n = input('Input number n = ');
e = input('Input number e = ');
logic = 0;
k = 2;

while logic == 0;
m = mod(n,k);

if m == 0;
p = k;
q = n/k;
k = 2;
logic = 1;
else
k = k+1;
end

end

P = sprintf('p = %i',p);
disp(P);
Q = sprintf('q = %i',q);
disp(Q);

phi = (p-1)*(q-1);

a_input = e;
b_input = phi;
logic = 0;
A_i = 0;
B_i = 1;
A_j = 1;
B_j = 0;
counter = 0;

if abs(a_input)>abs(b_input);
a = a_input;
a_0 = a_input;
b = b_input;
b_0 = b_input;
else
a = b_input;
a_0 = b_input;
b = a_input;
b_0 = a_input;
end

while logic ==0;
counter = counter + 1;
r = mod(a,b);
q = (a-r)/b;
A_k = q*A_j + A_i;
```

```
B_k = q*B_j + B_i;
if r ==0;
logic = 1;
else
A_i = A_j;
B_i = B_j;
A_j = A_k;
B_j = B_k;
a = b;
b = r;
end
end

sign = (-1)^counter;

if sign == 1;
u = B_j;
v = -A_j;
else
u = -B_j;
v = A_j;
end

X = sprintf('hcf(e, phi) = %i*(%i) + %i*(%i) = %i',a_0,u,b_0,v,b);
disp(X);

if abs(a_input)>abs(b_input);
d = mod(u,phi);
else
d = mod(v,phi);
end

D = sprintf('d = %i (mod %i)',d,n);
disp(D);
```

Appendix: Question 8

```
clear; clc;
format longg

n = input('Input number n = ');
%n = 937513;
e = input('Input number e = ');
%e = 638471;
c = input('Input message c = ');
logic = 0;
k = 2;

while logic == 0;
m = mod(n,k);

if m == 0;
p = k;
q = n/k;
k = 2;
logic = 1;
else
k = k+1;
end

end

P = sprintf('p = %i',p);
%disp(P);
Q = sprintf('q = %i',q);
%disp(Q);

phi = (p-1)*(q-1);

a_input = e;
b_input = phi;
logic = 0;
A_i = 0;
B_i = 1;
A_j = 1;
B_j = 0;
counter = 0;

if abs(a_input)>abs(b_input);
a = a_input;
a_0 = a_input;
b = b_input;
b_0 = b_input;
else
a = b_input;
a_0 = b_input;
b = a_input;
b_0 = a_input;
end

while logic ==0;
counter = counter + 1;
```

```
r = mod(a,b);
q = (a-r)/b;
A_k = q*A_j + A_i;
B_k = q*B_j + B_i;
if r ==0;
logic = 1;
else
A_i = A_j;
B_i = B_j;
A_j = A_k;
B_j = B_k;
a = b;
b = r;
end
end

sign = (-1)^counter;

if sign == 1;
u = B_j;
v = -A_j;
else
u = -B_j;
v = A_j;
end

X = sprintf('hcf(e, phi) = %i*(%i) + %i*(%i) = %i',a_0,u,b_0,v,b);
%disp(X);

if abs(a_input)>abs(b_input);
d = mod(u,phi);
else
d = mod(v,phi);
end

D = sprintf('d = %i (mod %i)',d,n);
%disp(D);

m = 1;
for k = 1:d;
m = mod( m*c, 937513);
end

C = sprintf('m = %i (mod %i)',m,n);
disp(C);
```

END