Project: 2.3 Public Key Cryptography

May 1, 2018

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).**

   There are two simple modifications that I implemented into my program. The first modification is that there is no need to see if our test number, n, is divisible by even numbers larger than 2. If it were the case that an even number larger than 2 divided n, then it must be the case that n was divisible by 2 in the first place.

   $$\text{If } 2k|n \text{ and } 2|2k \Rightarrow 2|n, \text{ where } k > 1,\ k \in \mathbb{N}$$

   The second simple modification was to realise that testing division by numbers greater than square root of n is unnecessary since factors of n come in pairs and the square root of n is the point past which the smaller of the pair of factors would have already been tested.

   The output from MATLAB of some 10 digit numbers with their elapsed time is as follows. When a number is not prime, the function outputs the first (prime) number that divides into our test number.

   >> `trialdivision1`(5915587277)
   It is Prime!
   Elapsed time is 0.004744 seconds.

   >> `trialdivision1`(4093082927)
   Not Prime
   k = 751
   Elapsed time is 0.000628 seconds.

   This is clearly a very fast method given they both execute within less than a hundredth of a second.

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?

   The output data of the MATLAB code for different 10 digit numbers is as follows:

   >> `primefactors2`(4093082927)
   Prime Factor: = 751
   Prime Factor: = 1511
   Prime Factor: = 3607
   Elapsed time is 0.001177 seconds.

   >> `primefactors2`(2343082940)
   Prime Factor: = 2
   Prime Factor: = 2
   Prime Factor: = 5
   Prime Factor: = 11
   Prime Factor: = 10650377
   Elapsed time is 0.000754 seconds.

   The complexity of my algorithm is the number of arithmetic operations needed in the worst case. The worst case scenario must involve the while loop. We can see that one iteration of the while loop contains an $O(1)$, constant, number of arithmetic operations since the MATLAB function, `mod(n,k)`, takes 3 arithmetic operations to calculate for any n and k.[1] Hence the number of while loops is proportional to the number of arithmetic operations and so the worst case scenario would be when the number of while loops executed is maximised.

   The worst case would be when n is the square of a single prime number. This is due to the largest number of while loops executed being obtained in the first instance of my program. To maximise this, we desire our program to test all numbers up to and including $\sqrt{n}$ to maximise the number of arithmetic operations. If n is prime, then this condition is satisifed; however if n is the sqaure of two prime numbers, then this condition is also satisfied and the program will run again to obtain the second factor. This means that when n is a square of a single prime number, we obtain the most arithmetic operations.

   In the case when $n = ab$ is the product of two prime numbers with one being slightly less than the $\sqrt{n}$, and one slightly more, say a and b respecitvely, we would get through the same number of arithmetic operations in the first while loop for n compared with n' when obtaining a (since $\sqrt{n} \approx \sqrt{n'}$ and so the

[1] https://uk.mathworks.com/help/symbolic/mupad_ref/mod.html

upper limit of the first while loop will be the same in both cases). However the second while loop for n will obtain b and will also result in the same number of arithmetic operations as for the second while loop for n' when obtaining a. This is because despite $n' < n$, $\sqrt{a} \approx \sqrt{b}$, and so the upper limit of the second while loop is also the same in both cases. Hence we can see that the worst case is when n is the product of a single prime number.

When n is the product of more than 2 prime factors, the complexity will be less than a number close to it that is the product of two prime numbers since we end up using more instances of the while loop for which the constraint reduces each time and so the complexity in turn will reduce.

Considering an extreme example, we can see how a square of a prime number requires more arithmetic operations than a prime number of similar magnitude using this algorithm. Consider $n_1 = 5646775167877$ and $n_2 = 5646768421849$:

>> `primefactors2`(5646775167877)
Prime Factor: = 5646775167877
Elapsed time is 0.155702 seconds.
count = 1188147

>> `primefactors2`(5646768421849)
Prime Factor: = 2376293
Prime Factor: = 2376293
Elapsed time is 0.165689 seconds.
count = 1188918

Although $n_1 > n_2$, it took a larger number of while loops, counted with the 'count' variable, and more time n_2's prime factorisation to be computed. This implies that when n is a square of a prime number, its complexity is slightly higher.

In general, if we consider the number of arithmetic operations in the case where n is the square of a single prime number, we obtain that the complexity is:

$$5 \times \frac{1}{2} \times n^{1/2} + 5 \times \frac{1}{2} \times n^{1/4} = O(\sqrt{n}),$$

where the 5 is number of arithmetic operations in each while loop and the $\frac{1}{2}$ accounts for the algorithm not testing even numbers apart from 2.

We can test to see if this agrees with the data obtained above. The number of while loops for n_2 will be equal to (not accounting for the factor of 5 in the above equation):

$$\begin{aligned}\frac{1}{2} \times n_2^{1/2} + \frac{1}{2} \times n_2^{1/4} &= \frac{1}{2} \times \sqrt{5646768421849} + \frac{1}{2} \times \sqrt[4]{5646768421849} \\ &= \frac{1}{2} \times 2376293 + \frac{1}{2} \times 1541.522\ldots \\ &= 1188917.261... \\ &\approx 1188918, \text{ which is what the count value states above for the number of while loops}\end{aligned}$$

Hence the complexity of my algorithm is $O(\sqrt{n})$.

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.**

>> `euclid3`(1 996 245 783, 192 784 863)
HCF = 3 = 1 996 245 783 × −11 108 123 + 192 784 863 × 115 022 224

>> `euclid3`(2 825 746 811, 758 295 345)
HCF = 1 = 2 825 746 811 × −28 353 319 + 758 295 345 × 105 657 118

>> `euclid3`(249 508 543 104, 338 063 357 376)
HCF = 46 656 = 338 063 357 376 × −356 165 + 249 508 543 104 × 482 574

>> `euclid3`(249 508 543 140, 338 063 357 367)
HCF = 9 = 338 063 357 367 × −8 485 693 393 + 249 508 543 140 × 11 497 409 916

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

We know that the linear congruence has a solution iff $\text{hcf}(a, m)|b$ (which we will prove below). First we will show that if $\text{hcf}(a, m) = 1$, then there exists a unique solution to the congruence. For the remainder of my answer, let $\text{hcf}(a, m) = h$.

If $h = 1$, we know that a has a multiplicative inverse, $a^{-1} \pmod{m}$ (see below for the explanation). We can then solve for x by multiplying both sides of the congruence by $a^{-1} \pmod{m}$ and obtaining an answer for $x \pmod{m}$.

$$a^{-1}ax \equiv a^{-1}b \pmod{m} \Rightarrow x \equiv a^{-1}b \pmod{m}$$

This is a unique solution, when $h = 1$, since if we assume there are two solutions x_1 and x_2, we can deduce that they are equal $\pmod{m}$.

$$ax_1 \equiv b \pmod{m} \text{ and } ax_2 \equiv b \pmod{m} \Rightarrow ax_1 \equiv ax_2 \pmod{m} \Rightarrow x_1 \equiv x_2 \pmod{m}$$

We can now prove the result that the linear congruence, $ax \equiv b \pmod{m}$, has a solution iff $\text{hcf}(a, m)|b$.

($\Rightarrow$): If the linear congruence has a solution, then $\exists x \pmod{m}$ such that $ax \equiv b \pmod{m}$ is satisfied. Hence it follows that $ax + mt = b$, for some $t \in \mathbb{Z}$. Since $h|a$ and $h|m$, $h|(ax + mt) \Rightarrow h|b$.

($\Leftarrow$): Assume $h|b$ and as before $h = \text{hcf}(a, m)$. We can then write $b = hb'$, $a = ha'$ and $m = hm'$. So $ax \equiv b \pmod{m} \Leftrightarrow ax - b = mt \Leftrightarrow ha'x - hb' = hm't \Leftrightarrow a'x - b' = m't \Leftrightarrow a'x \equiv b' \pmod{m'}$ for some $t \in \mathbb{Z}$. As we divided by the highest common factor of a and m, the $\text{hcf}(a', m') = 1$. Therefore the multiplicative inverse of a' exists (by the explanation below) and so there exists a (unique) solution $x \pmod{m'}$. This implies that there is a solution to $ax \equiv b \pmod{m}$, $x \pmod{m}$, since $m' < m$. □

In the case when $h \neq 1$ and $h|b$, we can divide a, b and m by h and obtain the equivalent linear congruence of:

$$\begin{aligned} \frac{ax}{h} &\equiv \frac{b}{h} \left(\text{mod } \frac{m}{h}\right) \\ a'x &\equiv b' \pmod{m'}. \end{aligned} \tag{1}$$

Since we divided by the highest common factor of a and m, $\text{hcf}(a', m') = 1$ and so the multiplicative inverse of a' exists. We can find x as above by multiplying both sides of the congruence by this inverse. To find all solutions in this case when $h \neq 1$, we can obtain our unique solution of (1), and then add multiples of m' to x until $m - (x + km') < 0$, for some $k \in \mathbb{Z}$.

To make it clearer, let $x = x_0 \pmod{m'}$, where $x_0 = (a')^{-1}b'$, be our unique solution to (1). Then all solutions to the congruence relation $\forall h$ are:

$$\begin{aligned} x &= x_0 \pmod{m} \\ x &= x_0 + m' \pmod{m} \\ x &= x_0 + 2m' \pmod{m} \\ &\vdots \\ x &= x_0 + (k-1)m' \pmod{m}, \end{aligned}$$

with the condition that $x_0 + (k-1)m' < m$ and $x_0 + (k)m' > m$, for some $k \in \mathbb{Z}$.

We can see that the solutions to this linear congruence depends solely on finding the multiplicative inverse of a or a'. The multiplicative inverse $\pmod{m}$ of, say, a exists iff $\text{hcf}(a, m) = 1$, i.e. a and m are coprime. This is true because:

($\Rightarrow$): Suppose the multiplicative inverse of a, $a^{-1} = u \pmod{m}$, exists. Then $au = 1 \pmod{m} \Leftrightarrow au + mv = 1$, for some $v \in \mathbb{Z}$. Since 1 can be written as a linear combination of a and $m \Rightarrow \text{hcf}(a, m) = 1$.

($\Leftarrow$): Suppose $\text{hcf}(a, m) = 1$, then by Euclid's Algorithm we can express 1 as a linear combination of a and m, $au + mv = 1$, for some $u, v \in \mathbb{Z}$. This implies that $au = 1 \pmod{m} \Rightarrow u = a^{-1} \pmod{m}$, the multiplicative inverse of a. □

As shown in question 3, Euclid's Algorithm can be used to represent the highest common factor of two numbers in terms of multiples of the original numbers. If we run Euclid's Algorithm on a and m, where the $\text{hcf}(a, m) = h = 1$, the algorithm states that $\exists u, v \in \mathbb{Z}$ such that

$$h = 1 = au + mv \Rightarrow au = 1 - mv \Rightarrow au = 1 \pmod{m} \Rightarrow u = a^{-1} \pmod{m},$$

where u and v are returned from the algorithm. Hence we can obtain our desired multiplicative inverse, $u \pmod{m}$, from Euclid's Algorithm and find our solution for $x \pmod{m}$.

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.**

   For the linear congruence $146295x \equiv 2017 \pmod{313567}$, the solution is:

   >> linearcongruence5(146295, 2017, 313567)
   $x = 267975 \pmod{313567}$

   For the linear congruence $93174x \equiv 2015 \pmod{267975}$, a solution does not exist.

   >> linearcongruence5(93174, 2015, 267975)
   This linear congruence has no solutions since the HCF of a and m does not divide b.

   For the linear congruence $113314x \equiv 2014 \pmod{660115}$, the (53) solutions are:

   >> linearcongruence5(113314, 2014, 660115)

$x = 11721 \pmod{660115}$	$x = 235911 \pmod{660115}$	$x = 460101 \pmod{660115}$
$x = 24176 \pmod{660115}$	$x = 248366 \pmod{660115}$	$x = 472556 \pmod{660115}$
$x = 36631 \pmod{660115}$	$x = 260821 \pmod{660115}$	$x = 485011 \pmod{660115}$
$x = 49086 \pmod{660115}$	$x = 273276 \pmod{660115}$	$x = 497466 \pmod{660115}$
$x = 61541 \pmod{660115}$	$x = 285731 \pmod{660115}$	$x = 509921 \pmod{660115}$
$x = 73996 \pmod{660115}$	$x = 298186 \pmod{660115}$	$x = 522376 \pmod{660115}$
$x = 86451 \pmod{660115}$	$x = 310641 \pmod{660115}$	$x = 534831 \pmod{660115}$
$x = 98906 \pmod{660115}$	$x = 323096 \pmod{660115}$	$x = 547286 \pmod{660115}$
$x = 111361 \pmod{660115}$	$x = 335551 \pmod{660115}$	$x = 559741 \pmod{660115}$
$x = 123816 \pmod{660115}$	$x = 348006 \pmod{660115}$	$x = 572196 \pmod{660115}$
$x = 136271 \pmod{660115}$	$x = 360461 \pmod{660115}$	$x = 584651 \pmod{660115}$
$x = 148726 \pmod{660115}$	$x = 372916 \pmod{660115}$	$x = 597106 \pmod{660115}$
$x = 161181 \pmod{660115}$	$x = 385371 \pmod{660115}$	$x = 609561 \pmod{660115}$
$x = 173636 \pmod{660115}$	$x = 397826 \pmod{660115}$	$x = 622016 \pmod{660115}$
$x = 186091 \pmod{660115}$	$x = 410281 \pmod{660115}$	$x = 634471 \pmod{660115}$
$x = 198546 \pmod{660115}$	$x = 422736 \pmod{660115}$	$x = 646926 \pmod{660115}$
$x = 211001 \pmod{660115}$	$x = 435191 \pmod{660115}$	$x = 659381 \pmod{660115}$
$x = 223456 \pmod{660115}$	$x = 447646 \pmod{660115}$	

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?

   My program, `primefactors2`(n), calculates the prime factorisation of any positive integer n. As shown in question 2, the complexity of the program is $O(\sqrt{n})$. Using a modified version of this, `primefactors6`(n), we can use it to find the prime factorisation of $n = pq$, where p and q are prime.

   `primefactors6`(n) only works out the first smallest prime number, WLOG $p < q$, that divides into n, and then outputs $q = \frac{n}{p}$ as well. So assuming a user inputs a semi-prime number, the output of the algorithm will indeed be the two prime numbers who's product is n.

   In practice, the values of p and q would be large and hence similar in magnitude, since RSA algorithm is more secure for large primes (for example two 100 digit prime numbers). This would mean that:

$$\begin{aligned} n = pq \Rightarrow \sqrt{n} &= \sqrt{pq} \approx \sqrt{p^2} \\ \sqrt{n} &\approx p \end{aligned} \tag{2}$$

   Hence the number of arithmetic operations required for this modified program would only involve one while loop and would be approximately:

$$\frac{4\sqrt{n}}{2} = 2\sqrt{n}, \tag{3}$$

   where 4 is the number of arithmetic operations in each while loop and the $\frac{\sqrt{n}}{2}$ is the approximate number of while loops executed (the 2 is to account for not testing division of even numbers). The exact number of arithmetic operations would be $2p$, however given (2), we can approximate this to (3) and so our estimation doesn't depend on the unknown p.

   We can improve on this answer if we maintain the assumption that p and q are similar in magnitude. We could modify the program further and test the division of numbers starting from $\sqrt{n}$ and going down to

2. `primefactors6_1`(n) does this and we can deduce that the number of arithmetic operations would be:

$$4 \times \left(\frac{\sqrt{n} - p}{2} \right) + C = 2(\sqrt{n} - p) + C, \tag{4}$$

where C is a constant depending on n and what branches of the if statements it takes. This doesn't assume that the values of p and q are similar in magnitude, as the number of arithmetic operations contains the variable p. However the number of arithmetic operations, (4), will tend to 0 as $|p| \to |q|$, which makes this program more efficient compared to `primefactors6`(n) for $|p| \approx |q|$.

If we are now given p and q, we know that $\varphi(n) = (p-1)(q-1)$. We know that there exists a d such that $ed \equiv 1 \pmod{\varphi(n)}$ which we can compute with the program and inputs `linearcongruence5`$(e, 1, \varphi(n))$. A good estimate of the number of arithmetic operations to compute d would be the complexity of the program, and hence the worst case given any a and b as inputs. The main problem in calculating this approximation lies in the stage where the HCF is computed. This is using Euclid's algorithm and so we must work out an approximation for the complexity of Euclid's algorithm, and then add up the remaining number of arithmetic operations required to compute d.

WLOG, $a > b$. The worst case scenario for Eulid's Algorithm is when the two input values, a and b, are both consecutive Fibonnaci numbers. This is a result of the following two propositions that I will prove. I will define F_i to be the i'th Fibonacci number, where $F_1 = 1$ and $F_2 = 1$ and $F_{i+2} = F_{i+1} + F_i$.

Proposition 1. *Applying Euclid's algorithm to F_{N+2} and F_{N+1}, the algorithm will terminate after precisely N steps.*

Proof. The first step of Euclid's algorithm is to decompose the larger number, F_{N+2}, in terms of a quotient, q_1, and remainder, r_1, when dividing by the smaller number, F_{N+1}. Then the next step would be to repeat this with F_{N+1} and r_1, since the HCF also divides r_1 if it divides the other two terms in equation in the first step. Below shows the all the steps of the algorithm, and we can clearly see that given the nature of the initial numbers, we know what $q_1 = 1$ and $r_1 = F_N$, by the definition of the Fibonacci numbers. Hence we can deduce the rest of the terms in the algorithm and see where it terminates.

$$\begin{aligned} F_{N+2} &= 1 \times F_{N+1} + F_N \\ F_{N+1} &= 1 \times F_N + F_{N-1} \\ &\vdots \\ F_4 &= 1 \times F_3 + F_2 \\ F_3 &= 2 \times F_2 + 0 \end{aligned}$$

The algorithm terminates when $r_i = 0$ for some $i \in \mathbb{N}^0$, and in this case it is $r_N = 0$. □

Proposition 2. *Given any $a, b \in \mathbb{N}^+$, $a > b$, such that when Euclid's algorithm is applied to them, the program terminates in N steps; then $a \geq F_{N+2}$ and $b \geq F_{N+1}$.*

Proof. By (strong) Induction.

Base Case: When N=1, this occurs when $b|a$ and so $a = q_1 \times b + 0$, for some $q_1 \in \mathbb{N}^+$. The smallest numbers a and b, $a > b$, such that running Euclid's Algorithm takes only one step would be when $a = 2$ and $b = 1$. Hence $a = 2 = F_{1+2}$ and $b = 1 = F_{1+1}$, which both satisfy the proposition above. Since these are the smallest numbers, we have found the lowest bound for when $N = 1$ and shown it to be the Fibonacci numbers, so the proposition is true for $N = 1$.

Assume true for $N \leq k$. Now let Euclid's algorithm with inputs a, b take $k+1$ steps. Hence the first two steps will be:

$$a = q_1 \times b + r_1 \tag{5}$$
$$b = q_2 \times r_1 + r_2, \tag{6}$$

where $q_i \in \mathbb{N}^+$. There are exactly k steps from equation (6) to the termination of the algorithm, and so by our induction hypothesis, we know that we can assume that $b \geq F_{k+2}$ and $r_1 \geq F_{k+1}$. Furthermore, we can assume that $r_2 \geq F_k$, since we assume our proposition holds for all $N \leq k$. We can then bound a and b below by using (5), (6) and the inequalities obtained.

$$\begin{aligned} a &= q_1 \times b + r_1 \geq b + r_1 \geq F_{k+2} + F_{k+1} = F_{k+3} \\ b &= q_2 \times r_1 + r_2 \geq r_1 + r_2 \geq F_{k+1} + F_k = F_{k+2} \end{aligned}$$

So $a \geq F_{k+3}$ and $b \geq F_{k+2}$, which implies that the proposition is true. □

Both of these propositions together indicate that the smallest two numbers that require N steps to run Euclid's Algorithm are indeed two consecutive Fibonacci numbers, namely F_{N+2} and F_{N+1}.

Now with these results, we can work out an upper bound of the complexity of Euclid's Algorithm, given input numbers $\varphi(n)$ and e. We can assume that $\varphi(n) > e$. Let N be the number of steps that Euclid's algorithm takes to terminate, with $\varphi(n)$ and e as inputs. Also let ϕ be defined as:

$$\phi = \frac{1+\sqrt{5}}{2}.$$

First I will show that $F_N \geq \phi^{N-2}$, for $N \geq 2$. We can prove this by induction. When $N = 2$, $F_2 = 1 \geq \phi^0 = 1$, and when $N = 3$, $F_3 = 2 \geq \phi^1 = 1.618\ldots$. Now assume this is true for $N = k$ and $N = k+1$, then we have that:

$$F_{N+2} = F_{N+1} + F_N \geq \phi^{N-1} + \phi^{N-2} = \phi^N, \text{ since } \phi^2 - \phi - 1 = 0.$$

Therefore the result $F_N \geq \phi^{N-2}$, for $N \geq 2$, holds. We can then consider the following two inequalities:

$$\begin{aligned} \varphi(n) &\geq F_{N+2} \geq \phi^N \\ \log_\phi\big(\varphi(n)\big) &\geq N \end{aligned} \tag{7}$$

and

$$\begin{aligned} e &\geq F_{N+1} \geq \phi^{N-1} \\ \log_\phi(e) + 1 &\geq N \end{aligned} \tag{8}$$

We have now bounded the number of steps, N, by each of the two inputs into Euclid's Algorithm. We can see that $\log_\phi(e) + 1 \geq \log_\phi\big(\varphi(n)\big)$ only if $e\phi \geq \varphi(n)$. In practice, $e << \varphi(n)$ hence the complexity will have the upper bound in equation (8). In the worst case scenario, when both inputs are Fibonnaci numbers, both of these upper bounds would be equal. Otherwise, since $e << \varphi(n)$, as the number of steps N increases, it will get closer to the upper limit $\log_\phi(e)+1$. Therefore Euclid's Algorithm has complexity:

$$O\big(\log_\phi(e)\big) = O\big(\log(e)\big)$$

My program, `linearcongruence5`$(e, 1, \varphi(n))$, will first calculate the HCF of the two numbers $\varphi(n)$ and e. As we have shown, the number of steps, N, for Euclid's Algorithm can be approximated by $N \approx \log_\phi(e)+1$. Each step in Euclid's Algorithm has 4 arithmetic operations to go through. So we can multiply our approximate step number by 4 for the total number of arithmetic operations that Euclid's Algorithm contributes.

Since we know that $\varphi(n)$ and e are coprime, we need not consider the complexity for the HCF$\neq 0$. My program then starts from $N-1$ and works backwards down to 4 to eventually obtain Bzout's identity (or it has separate cases if N is initially is 1,2 or 3). Each step going backwards takes 4 arithmetic operations, and since we are doing this from $N-1$ to 4 inclusive, we have a sub-total number of arithmetic operations: $4((N-1)-3)$.

The last step to calculate d takes 4 arithmetic operations and so the approximate overall total of arithmetic operations, A, to find d is:

$$\begin{aligned} A &= 4N + 4((N-1)-3) + 4 \\ A &= 4\big(\log_\phi(e)+1\big) + 4\Big(\big(\log_\phi(e)\big) - 3\Big) + 4 \\ A &= 8\log_\phi(e) - 4 = O(\log_\phi(e)) = O(\log(e)) \text{ as } N \to \infty \end{aligned}$$

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

   The value of d that solves the linear congruence $ed \equiv 1 \pmod{\varphi(n)}$, given the pairs of integers (n, e) below, is:

- $(1\,764\,053\,131, 103\,471)$

  `>> decryption7(1 764 053 131, 103 471)`
  $d = 191\,584\,927$
- $(1\,723\,466\,867, 692\,581\,937)$

  `>> decryption7(1 723 466 867, 692 581 937)`
  $d = 225\,248\,873$
- $(1\,805\,760\,301, 39\,871\,477)$

  `>> decryption7(1 805 760 301, 39 871 477)`
  $d = 1\,452\,797\,497$
- $(6\,734\,071\,952\,813, 2\,017)$

  `>> decryption7(6 734 071 952 813, 2017)`
  $d = 4\,073\,158\,775\,953$
- $(9\,976\,901\,028\,181, 837\,856\,358\,917)$

  `>> decryption7(9 976 901 028 181, 837 856 358 917)`
  $d = 3\,864\,734\,962\,285$
- $(1\,603\,982\,333, 927\,145)$

  `>> decryption7(1 603 982 333, 927 145)`
  $d = 1\,518\,941\,485$

   This value of d for each pair of integers (n, e) forms part of the decryption key. The whole decryption key for each pair would be (n, d). These two numbers n and d would allow someone to decrypt an encrypted message c by calculating:

$$m = c^d \pmod{n},$$

   where m is the original message.

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

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

   Please see the code at the end of my report named: `decryptionrsa8(n,e,c)`. This corresponds to the program required in this question. We can test this program by using an example. I will use the encoding given in the question, with public key (937513, 638471). We can use the program `encryption(n,e,m_i)` to encrypt two 3 letter words, say 'dog' and 'rat'.

   'dog' $\mapsto 041507 = 41507 = m_1$
   'rat' $\mapsto 180120 = m_2$

   Now we can encrypt these numbers using the encryption program above and obtain:

   `>> encryption(937513,638471,41507)`
   $c_1 = 874856$

   `>> encryption(937513,638471,180120)`
   $c_2 = 333686$

   We can now test the program `decryptionrsa8(n,e,c_i)` by inputting our values of c_1 and c_2.

   `>> decryptionrsa8(937513, 638471, 874856)`
   $m_1 = 41507$

   `>> decryptionrsa8(937513, 638471, 333686)`
   $m_2 = 180120$

   The output is numbers we started with, which shows that the program works correctly.

   The greatest number of digits that n can have for which my program can be trusted to work would be 16 digits since by default the MATLAB precision is 16 digits[2].

9. **I receive the encrypted message, with public key** $(937\,513, 638\,471)$,

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?

   Using my program `decryptionrsa8(937 513, 638 471, c)`, where c is the encrypted 6 digit input into the program, I can decode the message above. The decoded message is:

   `i'm not really in a cheese mood.  now that's what I call an egg sandwich.`

[2] https://uk.mathworks.com/help/symbolic/digits.html

```
function [outputArg1,outputArg2] = trialdivision1(n)
%Trial division - which divides every number prior to n to see if it is
%prime. Simple modification: we can see that if we check divisibilty
%by 2, we know it is not prime unless it is 2 itself. Also do not have to
%check divisibility for numbers greater than (sqrt(n) since they we get
%passed the point of prime factor pairs, the lower pair would have already
%been tested
%%ONLY CHECK PRIME NUMBERS...IF THEY DIVIDE INTO IT IS MOST EFFICIENT
format longg
tic;
k=2;
if n<=0
    disp('Please input a number greater than 0')
elseif n==1
    disp('Not Prime')
elseif n==2
    disp('It is Prime!')
else
%tests all the unique cases first
while k<=sqrt(n)
    if mod(n,k)==0
        disp('Not Prime')
        k
        toc
    return
    elseif k==2
        k=k+1;
    else
        k=k+2;
    end
end
disp('It is Prime!')
end
toc
end
```

```
function [outputArg1,outputArg2] = primefactors2(n)
%Prime factorisation - which divides every number prior to n to see if it is
%prime. Simple modification: we can see that if we check divisibilty
%by 2, we know it is not prime unless it is 2 itself.
%%ONLY CHECK PRIME NUMBERS...IF THEY DIVIDE INTO IT IS MOST EFFICIENT
format longg
tic;
k=2;
count=0;
if n<=1
    disp('Please input a number greater than 1 to obtain the prime factorisation')
end
while k<=sqrt(n)
    if mod(n,k)==0
        fprintf('Prime Factor: = %7.14g \\newline \n', k)
        n=(n/k);
        k=2;
    elseif k==2
        k=k+1;
    else
        k=k+2;
    end
    %count=count+1; can be used to count the number of while loops when
    %required.
end
fprintf('Prime Factor: = %7.14g \\newline \n', n)
toc
count;
end
```

```matlab
function [outputArg1,outputArg2] = euclid3(a,b)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
if a<=0 | b<=0
    disp('Please enter valid integers a and b, greater than 0')
    return
end
count=1;
    if a<b
        m=b;
        b=a;
        a=m;
    end
q=floor(a/b);
r=mod(a,b);
w(1,count)=1;
w(2,count)=a;
w(3,count)=q;
w(4,count)=b;
w(5,count)=r;
    while r~=0
        a=b;
        b=r;
        q=floor(a/b);
        r=mod(a,b);
        count=count+1;
        w(1,count)=1;
        w(2,count)=a;
        w(3,count)=q;
        w(4,count)=b;
        w(5,count)=r;
    end
w(3,:)=w(3,:)*(-1);
count
%cosider w anc how it changes as you go down the algorithm
if count > 3
        for k=count-1:-1:3
        w(:,k+1)=0;
        n=w(3,k);
        w(4,k)=w(2,k);
        w(3,k)=w(3,k-1)*n+w(1,k);
        w(2,k)=w(2,k-1);
        w(1,k)=w(1,k-1)*n;
        w(:,k-1)=w(:,k);
        k=k+1;
        end
    k=2;
    w(:,k+1)=0;
    n=w(3,k);
    w(4,k)=w(2,k);
    w(3,k)=w(3,k-1)*n+w(1,k);
    w(2,k)=w(2,k-1);
    w(1,k)=w(1,k-1)*n;
    w(:,k-1)=w(:,k);
elseif count==1
    w(:,2)=w(:,1);
    w(3,2)=w(3,2)+1;
    w(5,2)=w(4,2);
elseif count==2
    w(:,2)=w(:,1);
elseif count==3
    k=2;
    w(:,k+1)=0;
    n=w(3,k);
    w(4,k)=w(2,k);
    w(3,k)=w(3,k-1)*n+w(1,k);
    w(2,k)=w(2,k-1);
    w(1,k)=w(1,k-1)*n;
    w(:,k-1)=w(:,k);
end
HCF=w(5,2);
a=w(2,2);
u=w(1,2);
b=w(4,2);
v=w(3,2);
fprintf('HCF = $ %3g = %10.f \\times %10.f + %10.f \\times %10.f $ \\newline \n',HCF,a,u,b,v)
```

```
end
```

```
function [outputArg1,outputArg2] = linearcongruence5(a,b,m)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
if a<=0 | m<=0
    disp('Please enter valid integers a and b, greater than 0')
    return
end
b=mod(b,m);
a_0=a;
m_0=m;
count=1;
    if a<m
        s=m;
        m=a;
        a=s;
    end
q=floor(a/m);
r=mod(a,m);
w(1,count)=1;
w(2,count)=a;
w(3,count)=q;
w(4,count)=m;
w(5,count)=r;
    while r~=0
        a=m;
        m=r;
        q=floor(a/m);
        r=mod(a,m);
        count=count+1;
        w(1,count)=1;
        w(2,count)=a;
        w(3,count)=q;
        w(4,count)=m;
        w(5,count)=r;
    end
HCF=w(4,count);

if mod(b,HCF)~=0
                disp('This linear congruence has no solutions since the HCF of a and m does not divide b.')
                return
elseif HCF==1
        w(3,:)=w(3,:)*(-1);
            if count > 3
                    for k=count-1:-1:3
                    w(:,k+1)=0;
                    n=w(3,k);
                    w(4,k)=w(2,k);
                    w(3,k)=w(3,k-1)*n+w(1,k);
                    w(2,k)=w(2,k-1);
                    w(1,k)=w(1,k-1)*n;
                    w(:,k-1)=w(:,k);
                    k=k+1;
                    end
                k=2;
                w(:,k+1)=0;
                n=w(3,k);
                w(4,k)=w(2,k);
                w(3,k)=w(3,k-1)*n+w(1,k);
                w(2,k)=w(2,k-1);
                w(1,k)=w(1,k-1)*n;
                w(:,k-1)=w(:,k);
            elseif count==1
                w(:,2)=w(:,1);
                w(3,2)=w(3,2)+1;
                w(5,2)=w(4,2);
            elseif count==2
                w(:,2)=w(:,1);
            elseif count==3
                k=2;
                w(:,k+1)=0;
                n=w(3,k);
                w(4,k)=w(2,k);
                w(3,k)=w(3,k-1)*n+w(1,k);
                w(2,k)=w(2,k-1);
                w(1,k)=w(1,k-1)*n;
```

```
                w(:,k-1)=w(:,k);
            end
            HCF=w(5,2);
            if a_0<m_0
                a=w(4,2);
                u=w(3,2);
                m=w(2,2);
                v=w(1,2);
            else
                a=w(2,2);
                u=w(1,2);
                m=w(4,2);
                v=w(3,2);
            end
else
                count=1;
                a=w(2,1)/HCF;
                b=b/HCF;
                m_1=w(4,1)/HCF;
                q=floor(a/m_1);
            r=mod(a,m_1);
            w(1,count)=1;
            w(2,count)=a;
            w(3,count)=q;
            w(4,count)=m_1;
            w(5,count)=r;
                while r~=0
                    a=m_1;
                    m_1=r;
                    q=floor(a/m_1);
                    r=mod(a,m_1);
                    count=count+1;
                    w(1,count)=1;
                    w(2,count)=a;
                    w(3,count)=q;
                    w(4,count)=m_1;
                    w(5,count)=r;
                end
      w(3,:)=w(3,:)*(-1);
            if count > 3
                    for k=count-1:-1:3
                    w(:,k+1)=0;
                    n=w(3,k);
                    w(4,k)=w(2,k);
                    w(3,k)=w(3,k-1)*n+w(1,k);
                    w(2,k)=w(2,k-1);
                    w(1,k)=w(1,k-1)*n;
                    w(:,k-1)=w(:,k);
                    k=k+1;
                    end
                k=2;
                w(:,k+1)=0;
                n=w(3,k);
                w(4,k)=w(2,k);
                w(3,k)=w(3,k-1)*n+w(1,k);
                w(2,k)=w(2,k-1);
                w(1,k)=w(1,k-1)*n;
                w(:,k-1)=w(:,k);
            elseif count==1
                w(:,2)=w(:,1);
                w(3,2)=w(3,2)+1;
                w(5,2)=w(4,2);
            elseif count==2
                w(:,2)=w(:,1);
            elseif count==3
                k=2;
                w(:,k+1)=0;
                n=w(3,k);
                w(4,k)=w(2,k);
                w(3,k)=w(3,k-1)*n+w(1,k);
                w(2,k)=w(2,k-1);
                w(1,k)=w(1,k-1)*n;
                w(:,k-1)=w(:,k);
            end
            HCF=w(5,2);
                if a_0<m_0
```

```
                    a=w(4,2);
                    u=w(3,2);
                    m_1=w(2,2);
                    v=w(1,2);
                else
                    a=w(2,2);
                    u=w(1,2);
                    m_1=w(4,2);
                    v=w(3,2);
                end
            m=m_1;
end
ainv=u;
x=mod(ainv*b,m);
while m_0-x>0
fprintf('$ x = %4.f \\Mod{%4.f}$ \\newline \n',x,m_0)
    x=x+m;
end
end
```

```
function [outputArg1,outputArg2] = primefactors6(n)
%Prime factorisation - for n=pq, where p<0 and p is prime
%%ONLY CHECK PRIME NUMBERS...IF THEY DIVIDE INTO IT IS MOST EFFICIENT
format longg
tic;
count=0;
k=2;
if n<=1
    disp('Please input a number greater than 1 to obtain the pq prime factorisation')
    return
end
while k<=sqrt(n)
    if mod(n,k)==0
        fprintf('p = %10.f \\newline \n', k)
        fprintf('q = %10.f \\newline \n', n/k)
        toc
        count;
        return
    elseif k==2
        k=k+1;
    else
        k=k+2;
    end
    %count=count+1;
end
k=n;
fprintf('p = %10.f \\newline \n', k)
fprintf('q = %10.f \\newline \n', 1)
end
```

```
function [outputArg1,outputArg2] = primefactors6_1(n)
%Prime factorisation - for n=pq, where p<0 and p is prime
%%ONLY CHECK PRIME NUMBERS...IF THEY DIVIDE INTO IT IS MOST EFFICIENT
format longg
tic;
count=0;
if n<=1
    disp('Please input a number greater than 1 to obtain the pq prime factorisation')
    return
end
if mod(floor(sqrt(n)),2)==0
    k=floor(sqrt(n)-1);
else
    k=floor(sqrt(n));
end
while k>2
    if mod(n,k)==0
        fprintf('p = %10.f \\newline \n', k)
        fprintf('q = %10.f \\newline \n', n/k)
        toc
        count;
        return
    else
        k=k-2;
    end
    %count=count+1;
end
if mod(n,2)==0
    k=2
    fprintf('p = %10.f \\newline \n', k)
    fprintf('q = %10.f \\newline \n', n/k)
    toc
    count;
else
    k=n
    fprintf('p = %10.f \\newline \n', k)
    fprintf('q = %10.f \\newline \n', n/k)
    toc
    count;
end
```

```matlab
function [outputArg1,outputArg2] = decryption7(n,e)
%Prime factorisation first of n=pq, where p<0 and p is prime
%Then we find phi(n), HCF(phi(n),e) and find the inverse of e mod(phi(n))
%%ONLY CHECK PRIME NUMBERS...IF THEY DIVIDE INTO IT IS MOST EFFICIENT?
format longg
tic;
count=0;
k=2;
if n<=1
    disp('Please input a number n greater than 1 to obtain the pq prime factorisation')
    return
end
if e<=0
    disp('Please input a number e greater than 0')
    return
end
while k<=sqrt(n)
    if mod(n,k)==0
        p=k;
        q=n/k;
        break
    elseif k==2
        k=k+1;
    else
        k=k+2;
    end
end
if k>=(sqrt(n))
  disp('n is a prime number, please enter another value for n such that n=pq')
  return
end
phi=(p-1)*(q-1);

            a_0=e;
            m_0=phi;
            count=1;
                if e<phi
                    s=phi;
                    phi=e;
                    e=s;
                end
            q=floor(e/phi);
            r=mod(e,phi);
            w(1,count)=1;
            w(2,count)=e;
            w(3,count)=q;
            w(4,count)=phi;
            w(5,count)=r;
                while r~=0
                    e=phi;
                    phi=r;
                    q=floor(e/phi);
                    r=mod(e,phi);
                    count=count+1;
                    w(1,count)=1;
                    w(2,count)=e;
                    w(3,count)=q;
                    w(4,count)=phi;
                    w(5,count)=r;
                end
            HCF=w(4,count);

            if mod(1,HCF)~=0
                            disp('phi(n) and e are not coprime')
                            return
            elseif HCF==1
                    w(3,:)=w(3,:)*(-1);
                        if count > 3
                                for k=count-1:-1:3
                                w(:,k+1)=0;
                                n=w(3,k);
                                w(4,k)=w(2,k);
                                w(3,k)=w(3,k-1)*n+w(1,k);
                                w(2,k)=w(2,k-1);
                                w(1,k)=w(1,k-1)*n;
                                w(:,k-1)=w(:,k);
```

```
                                k=k+1;
                                end
                            k=2;
                            w(:,k+1)=0;
                            n=w(3,k);
                            w(4,k)=w(2,k);
                            w(3,k)=w(3,k-1)*n+w(1,k);
                            w(2,k)=w(2,k-1);
                            w(1,k)=w(1,k-1)*n;
                            w(:,k-1)=w(:,k);
                        elseif count==1
                            w(:,2)=w(:,1);
                            w(3,2)=w(3,2)+1;
                            w(5,2)=w(4,2);
                        elseif count==2
                            w(:,2)=w(:,1);
                        elseif count==3
                            k=2;
                            w(:,k+1)=0;
                            n=w(3,k);
                            w(4,k)=w(2,k);
                            w(3,k)=w(3,k-1)*n+w(1,k);
                            w(2,k)=w(2,k-1);
                            w(1,k)=w(1,k-1)*n;
                            w(:,k-1)=w(:,k);
                        end
                        HCF=w(5,2);
                        if a_0<m_0
                            e=w(4,2);
                            u=w(3,2);
                            phi=w(2,2);
                            v=w(1,2);
                        else
                            e=w(2,2);
                            u=w(1,2);
                            phi=w(4,2);
                            v=w(3,2);
                        end
            end
d=mod(u,m_0)
end
```

```
function [outputArg1,outputArg2] =decryptionrsa8(n,e,c)
%Prime factorisation first of n=pq, where p<0 and p is prime
%Then we find phi(n), HCF(phi(n),e) and find the inverse of e mod(phi(n))
%Then we raise c^d (mod(n)) and find m (our message)
%%ONLY CHECK PRIME NUMBERS...IF THEY DIVIDE INTO IT IS MOST EFFICIENT?
format longg
tic;
count=0;
k=2;
if n<=1
    disp('Please input a number n greater than 1 to obtain the pq prime factorisation')
    return
end
if e<=0
    disp('Please input a number e greater than 0')
    return
end
while k<=sqrt(n)
    if mod(n,k)==0
        p=k;
        q=n/k;
        break
    elseif k==2
        k=k+1;
    else
        k=k+2;
    end
end
if k>=(sqrt(n))
  disp('n is a prime number, please enter another value for n such that n=pq')
  return
end
n_0=n;
phi=(p-1)*(q-1);

            a_0=e;
            m_0=phi;
            count=1;
                if e<phi
                    s=phi;
                    phi=e;
                    e=s;
                end
            q=floor(e/phi);
            r=mod(e,phi);
            w(1,count)=1;
            w(2,count)=e;
            w(3,count)=q;
            w(4,count)=phi;
            w(5,count)=r;
                while r~=0
                    e=phi;
                    phi=r;
                    q=floor(e/phi);
                    r=mod(e,phi);
                    count=count+1;
                    w(1,count)=1;
                    w(2,count)=e;
                    w(3,count)=q;
                    w(4,count)=phi;
                    w(5,count)=r;
                end
            HCF=w(4,count);

            if mod(1,HCF)~=0
                            disp('phi(n) and e are not coprime')
                            return
            elseif HCF==1
                    w(3,:)=w(3,:)*(-1);
                        if count > 3
                                for k=count-1:-1:3
                                w(:,k+1)=0;
                                n=w(3,k);
                                w(4,k)=w(2,k);
                                w(3,k)=w(3,k-1)*n+w(1,k);
                                w(2,k)=w(2,k-1);
```

```
                                w(1,k)=w(1,k-1)*n;
                                w(:,k-1)=w(:,k);
                                k=k+1;
                                end
                            k=2;
                            w(:,k+1)=0;
                            n=w(3,k);
                            w(4,k)=w(2,k);
                            w(3,k)=w(3,k-1)*n+w(1,k);
                            w(2,k)=w(2,k-1);
                            w(1,k)=w(1,k-1)*n;
                            w(:,k-1)=w(:,k);
                        elseif count==1
                            w(:,2)=w(:,1);
                            w(3,2)=w(3,2)+1;
                            w(5,2)=w(4,2);
                        elseif count==2
                            w(:,2)=w(:,1);
                        elseif count==3
                            k=2;
                            w(:,k+1)=0;
                            n=w(3,k);
                            w(4,k)=w(2,k);
                            w(3,k)=w(3,k-1)*n+w(1,k);
                            w(2,k)=w(2,k-1);
                            w(1,k)=w(1,k-1)*n;
                            w(:,k-1)=w(:,k);
                        end
                        HCF=w(5,2);
                        if a_0<m_0
                            e=w(4,2);
                            u=w(3,2);
                            phi=w(2,2);
                            v=w(1,2);
                        else
                            e=w(2,2);
                            u=w(1,2);
                            phi=w(4,2);
                            v=w(3,2);
                        end
            end
d=mod(u,m_0);

c=mod(c,n_0);
r=0;
product=1;
odd=[1];
while d>1
        if mod(d,2)~=0
            d=d-1;
            r=r+1;
            odd(1,r)=c;
            c=mod(c^2,n_0);
            d=d/2;
        else
            c=mod(c^2,n_0);
            d=d/2;
        end
product=mod(prod(odd),n_0);
odd=[product];
r=1;
end
product;
m=mod(c*product,n_0)
end
```

```
function [outputArg1,outputArg2] = encryption(n,e,m)
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
n_0=n;
m=mod(m,n_0);
r=0;
evencounter=0;
product=1;
odd=[1];
while e>1
        if mod(e,2)~=0
            e=e-1;
            r=r+1;
            odd(1,r)=m;
            m=mod(m^2,n_0);
            e=e/2;
        else
            m=mod(m^2,n_0);
            e=e/2;
        end
product=mod(prod(odd),n_0);
odd=[product];
r=1;
end
product;
c=mod(m*product,n_0)
end
```