Algorithms - Solution 10

1. To find the root of c we perform a binary search. We compute $\left(\frac{c}{2}\right)^2$, if it is greater than c we continue to search in the first half, if it is smaller than c we continue to search in the second half. At each iteration we need to find the mid-point of the interval we are currently checking and to square it. Both operations can be done in time polynomial in $O(\log c)$. The number of iteration is $O(\log c)$.

2. (a) $(0, 4, 3)$
   (b) $n_1 = 3$, $n_2 = 5$, $n_3 = 8$, $n = 120$.
   $a_1 = 2$, $a_2 = 4$, $a_3 = 0$.
   $m_1 = 40$, $m_2 = 24$, $m_3 = 15$.
   $m_1^{-1}(\mod n_1) = 1$, $m_2^{-1}(\mod n_2) = 4$, $m_3^{-1}(\mod n_3) = 7$.
   $c_1 = 40$, $c_2 = 96$, $c_3 = 105$.
   $a = (2 \cdot 40 + 4 \cdot 96 + 0) \mod 120 = 104$.

3. Recall that $ed = 1 (\mod \phi(n))$. Also $\phi(n) = (p-1)(q-1)$ and $n = pq$. Since $e = 3$, and $d < \phi(n)$, we know that $ed = \phi(n) + 1$ or $ed = 2\phi(n) + 1$. Let us assume it is the first case (we do the same for the second case). Then we have the following equations in the variables p, q: $pq = n$ and $ed = pq - p - q + 1 = n - p - q + 1$. So we have $p = n - ed - q + 1$. We substitute this in the first equation, and get a quadratic equation in the variable q. We solve it, if one of its solutions is a factor of n then we are done, if not we do the same for the case $ed = 2\phi(n) + 1$. One of the two sets of equations must have the factor q as its solution.

4. Suppose that A is a procedure that can decrypt $1/100$ of the messages that Alice encrypts. We show a procedure that decrypts every message with high probability: Let $c(M_1)$ be an encrypted message and we want to compute M_1. We choose randomly a message M_2 and compute $gcd(M_2, n)$ (recall that n is public). If it is not 1, then we have succeeded to factor n and we can decrypt every message. Otherwise look at the message $M = M_1 M_2$. since M_2 is chosen uniformly from all the messages that have $gcd(M_2, n) = 1$, and since $gcd(M_1, n) = 1$, M is uniformly distributed over all the elements of the multiplicative group Z_n^* which is the space of all messages. Therefore with probability $1/100$ A can decrypt $c(M)$. We now use the fact that $c(M) = c(M_1)c(M_2)$. we compute $c(M_2)$ and multiply it by $c(M_1)$ we now have $c(M)$. we now give $c(M)$ to A. Suppose A computes M for us, so

we divide it by M_2 and retrieve M_1. We have shown that with probability $1/100$ we have succeeded to decrypt every message. If we repeat it many times (say, 1000) we succeed with very high probability.

5. (a) We have seen that computing GCD takes time which is polynomial in the length of the binary representation of the numbers. The same is true for squaring and computing remainders. Finally, by our assumption A runs in polynomial-time.

   (b) We want to show that given an integer y, the algorithm distinguishes between the case that y is prime and the case that y has at least two distinct prime factors both strictly larger than 2.

   Case 1: y is a prime. In this case $gcd(z, y) = 1$, and since Z_y is a field, there are two roots to $z^2 \mod y$: z and $-z$ (which is the integer $y - z$). A will return one of the two and we will declare correctly that y is a prime number (with probability 1).

   Case 2: y has two distinct prime factors. If $gcd(z, y) \neq 1$ then we have found a factor of y and we can declare that it is not a prime. Otherwise, using the same analysis that we did for Rabin's encryption scheme, we have that $z^2 \mod y$ has at least four different square roots ($\mod y$). Let R be the set of roots returned by A, and $-R$ be the set of their negatives (i.e. if $z \in R$, $y - z \in -R$). By our argument, $R \cup -R$ includes at most $1/2$ of the elements in Z_y^* (which is the set from which z comes, because $gcd(z, y) = 1$). Whenever we choose $z \notin R \cup -R$, we will get back from A a number that is not z or $y - z$ and we will correctly declare that y is not a prime. This happens with probability at least $1/2$.

   (c) We repeat the above algorithm $\log 1/\delta$ times, where each time z is chosen independently. If the above algorithm always answers "prime" we will say that y is prime. If in at least one iteration we get "not a prime" we will declare that y is not a prime. By the analysis above, if y is prime we will always say the correct answer. Otherwise, in each iteration we have a probability at most 1/2 to give the wrong answer (i.e. say that y is a prime). So the probability to give the wrong answers in all the iterations is at most $(1/2)^{\log 1/\delta} = \delta$.