Chapter 1–Solutions

February 2, 2007

1.1. A single digit number is at most $b-1$, therefore the sum of any three such numbers is at most $3b-3$. On the other hand, a two-digit number can be as large as b^2-1. It is enough to show that $b^2-1 \geq 3b-3$. Indeed, $b^2-1-3b-3 = (b-1)\cdot(b-2)$, which is ≥ 0 for $b \geq 2$.

1.2. For a number N we need $\lceil \log_2(N+1) \rceil$ binary digits and $\lceil \log_{10}(N+1) \rceil$ decimal digits. By the logarithm conversion formula: $\lceil \log_{10}(N+1) \rceil = \lceil \log_2(N+1) \cdot \log_2(10) \rceil \leq \lceil \log_2(N+1) \rceil \cdot \lceil \log_2(10) \rceil = 4 \cdot \lceil \log_2(N+1) \rceil$. For very large numbers, $\lceil \log_2(N+1) \cdot \log_2(10) \rceil \simeq \log_2(N+1) \cdot \log_2(10)$ so the required ratio is approximately equal to $\log_2(10) \simeq 3.8$.

1.3. The minimum depth of a d-ary tree is achieved when all nodes have precisely d children. In that case, the depth is $\log_d(n)$. For any d-ary tree, we have depth $D \leq \log_d(n) = \frac{\log(n)}{\log(d)} = \Omega(\frac{\log(n)}{\log(d)})$

1.4. We can lower bound $n!$ as

$$\underbrace{\left(\frac{n}{2}\right) \cdot \cdots \cdot \left(\frac{n}{2}\right)}_{\frac{n}{2} \text{ terms}} \leq 1 \cdot 2 \cdot 3 \cdot \cdots \cdot \left(\frac{n}{2}\right) \cdot \underbrace{\left(\frac{n}{2}+1\right) \cdot \cdots \cdot n}_{\frac{n}{2} \text{ terms}}$$

and upper bound it as

$$\underbrace{1 \cdot 2 \cdot 3 \cdot \cdots \cdot n}_{n \text{ terms}} \leq \underbrace{n \cdot \cdots \cdot n}_{n \text{ terms}}$$

Hence,

$$\begin{aligned} \left(\frac{n}{2}\right)^{\frac{n}{2}} &\leq n! \leq n^n, \\ \frac{n}{2}\log(\frac{n}{2}) &\leq \log(n!) \leq n \log n, \\ \frac{1}{2}(n \log n - n) &\leq \log(n!) \leq n \log n. \end{aligned}$$

1.5. Upper bound:

$$\begin{aligned} \sum_{i=1}^{n} \frac{1}{i} =& 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots \leq \\ \leq& 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \cdots + \underbrace{\frac{1}{2^k} + \cdots + \frac{1}{2^k}}_{2^k \text{ terms}} = \\ =& \underbrace{1 + 1 + \cdots + 1 + 1}_{O(\log n) \text{ terms}} = \\ =& O(\log n) \end{aligned}$$

Lower bound:

$$\begin{aligned}\sum_{i=1}^{n}\frac{1}{i} =& 1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots \geq \\ \geq& 1+\frac{1}{2}+\frac{1}{4}+\frac{1}{4}+\frac{1}{8}+\frac{1}{8}+\frac{1}{8}+\frac{1}{8}\cdots+\underbrace{\frac{1}{2^k}+\cdots+\frac{1}{2^k}}_{2^{k-1}\text{ terms}} = \\ =& 1+\underbrace{\frac{1}{2}+\frac{1}{2}+\cdots+\frac{1}{2}+\frac{1}{2}}_{\Omega(\log n)\text{ terms}} = \\ =& \Omega(\log n)\end{aligned}$$

1.6. First observe that multiplication of a number N by a one-digit binary number b results in the number 0 if $b = 0$ and N if $b = 1$. Also, multiplication of a binary number by 2^k for any power of 2 results in shifting N to the left k times and adding k digits equal to 0 at the end.
Let N, M the numbers we need to multiply. Assume M is a k-digit binary number. Write $M = b_0 + 2b_1 + \cdots + 2^k b_k$. Then $N \cdot M = N \cdot (b_0 + 2b_1 + \cdots + 2^k b_k) = Nb_0 + 2Nb_1 + \cdots + 2^k Nb_k$. If we consider the above observations, this expression is the same as the one illustrated in page 24 for the grade-school multiplication.

1.7. Assume we want to multiply the n-bit number x with the m-bit number y. The algorithm must terminate after m recursive calls, because at each call y is halved (the number of digits is decreased by one). Each recursive call requires a division by 2, a check for even-odd, a multiplication by 2 (all of those take constant time) and a possible addition of x to the current result (which takes $O(n)$ time) so total $O(m \cdot n)$ time.

1.8. We will first prove the correctness of the algorithm by induction on the number x. Base case is when $x = 0$ where correctness is obvious. Assume the algorithm returns correct values of q', r' so that $\lfloor \frac{x}{2} \rfloor = q'y + r'$. If x is even then $x = 2 \cdot \lfloor \frac{x}{2} \rfloor = 2q' + 2r'$. If $2r' \geq y$ then $x = (2q' + 1)y + (2r' - y)$, otherwise $x = 2q'y + 2r'$, which is exactly what the last call of the algorithm returns. If x is odd, then $x = 2 \cdot \lfloor \frac{x}{2} \rfloor + 1 = 2q' + 2r' + 1$. Again, according to $2r' + 1 \geq y$ or not it is easy to see that the algorithm returns the correct values of q and r.
Now, considering the running time : there are n recursive calls (each time x is halved, so we have one bit less). Each call requires two multiplications by 2, a check of even or odd, a possible addition of 1 (all of those take constant time) and possibly a subtraction of y which takes $O(n)$ time. The total running time is $O(n^2)$.

1.9. $x \equiv x' \mod N$ implies that N divides $x - x'$, similarly N divides $y - y'$. Consider the difference $D_1 = x + y - x' - y' = (x - x') + (y - y')$. We easily conclude that N divides D_1 therefore $x + y \equiv x' + y' \mod N$.
Similarly, define $D_2 = xy - x'y' = xy - xy' + xy' - x'y' = x(y - y') + (x - x')y$. We can easily see that N divides D_2 so $xy \equiv x'y' \mod N$.

1.10. $a \equiv b \mod N$ means N divides $a - b$. Since M divides N, then M also divides $a - b$ which leads to the conclusion $a \equiv b \mod M$.

1.11. We first observe that $35 = 5 \cdot 7$ and by Fermat's Little Theorem and simple algebra, $a^{(5-1)\cdot(7-1)} = a^{4 \cdot 6} = a^2 4 \equiv 1 \mod 35$, for all $1 \leq a < 35$. Therefore, $4^{1536} = 4^{24 \cdot 64} \equiv 1 \mod 35$ and $9^{4824} = 9^{24 \cdot 201} \equiv 1 \mod 35$. We conclude that $4^{1536} \equiv 9^{4824} \mod 35$ so the difference is divisible by 35.

1.12. $2^{2^{2006}} = 2^{2 \cdot 2^{2005}} = 4^{2^{2005}} \equiv 1 \mod 3$.

1.13. Since 31 is a prime number, by Fermat's Little Theorem we get $a^{30} \equiv 1 \mod 31$ for all $1 \leq a < 31$. Therefore $5^{30000} \equiv 1 \mod 31$. On the other hand, $6^{123456} = 6^{123450+6} = 6^6 = 5^3 = 125 \mod 31 \equiv 1 \mod 31$. So the given difference is a multiple of 31.

1.14. We will assume that the running time of multiplying n-bit numbers is $M(n)$. According to problem 0.4 in Chapter 0, fib3 involves $O(\log n)$ arithmetic operations (multiplications). Since we are working in arithmetic $\mod p$, each intermediate result will be $\log p$ bits long, therefore the total running time of fib3 is $O(\log n \cdot M(\log p))$.

1.15. $ax \equiv bx \mod c \Rightarrow (a-b)x \equiv 0 \mod c$. If $\gcd(x, c) = 1$ then $a \equiv b \mod c$ for all a, b. Conversely, if $ax \equiv bx \Rightarrow a \equiv b \mod c$ for all a, b then assume $c = nm$. Take a, b such that $a - b \equiv n \mod c \neq 0 \mod c$. For x a multiple of m we can still get $ax \equiv bx$ without having the second assertion. Therefore, x cannot have common factors with c.

1.16. Let $b = 15$. The algorithm of repeated squaring will calculate a, a^2, a^4, a^8 and then calculate $a^{15} = a \cdot a^2 \cdot a^4 \cdot a^8$, with a total of 6 multiplications.
Now, consider the following algorithm: first compute $a^3 = a \cdot a \cdot a$, then compute $a^6 = a^3 \cdot a^3$ and finally $a^{12} = a^6 \cdot a^6$. Calculate $a^{15} = a^{12} \cdot a^3$, with a total of 5 multiplications.

1.17. The iterative algorithm multiplies each time x into the result of the previous iteration. It performs $y-1$ iterations in total. In each iteration i, the size of the intermediate result m_i is at most $m_{i-1} + n$ bits long. Therefore the total running time is $O(n^2+2n^2+3n^2+\cdots+(y-1)n^2) = O(n^2 \cdot \frac{y(y-1)}{2}) = O(n^2 \cdot y^2)$. The recursive algorithm has $O(\log y)$ iterations and during the i-th iteration, multiplies two 2^{i-1} - bit long numbers. The total running time is $O(n^2+2^2n^2+2^4n^2+\cdots+2^{\lfloor \log y \rfloor - 1}n^2) = O(n^2 \cdot \frac{(2^2)^{\lfloor \log y \rfloor}-1}{3}) = O(n^2 \cdot y^2)$. We conclude that the two algorithms have running times of the same order.

1.18. First, we compute the gcd by finding the factorization of each number: $210 = 2 \cdot 3 \cdot 5 \cdot 7$ and $588 = 2^2 \cdot 3 \cdot 7^2$ therefore $\gcd(210, 588) = 2 \cdot 3 \cdot 7 = 42$.
Next, using Euclid's algorithm : $\gcd(210, 588) = \gcd(210, 168) = \gcd(168, 42) = 42$.

1.19. We can show this by induction on n. For $n = 1$, $\gcd(F_1, F_2) = \gcd(1, 1) = 1$. Now say that the inductive hypothesis is true for all $n \leq k$. This implies that for $n = k + 1$

$$\gcd(F_{k+1}, F_{k+2}) = \gcd(F_{k+1}, F_{k+2} - F_{k+1}) = \gcd(F_{k+1}, F_k) = 1$$

Hence, the statement is true for all $n \geq 1$.

1.20. (i) $(20)^{-1} = 4 \mod 79$
(ii) $(3)^{-1} = 21 \mod 62$
(iii) $\gcd(21, 91) = 7$. Hence, no inverse exists!
(iv) $(5)^{-1} = 14 \mod 23$

1.21. A number x has an inverse modulo 11^3 if and only if $\gcd(x, 11^3) = 1$. For $\gcd(x, 11^3) \neq 1$, either $x = 0$ or shares a common factor with 11^3. But the only factors of 11^3 are $11, 121$ and 1331. Thus, a nonzero number does not have an inverse if and only if it is a multiple of 11. The number of multiples of 11 between 1 and 1330 is $1331/11 - 1 = 120$ (because we only consider numbers upto 1330). Thus, the number of numbers that have an inverse is $1331 - 120 - 1 = 1210$.

1.22. Since a has an inverse $\mod b$ it means that a, b are coprime so b also has an inverse $\mod a$.

1.23. Suppose x_1 and x_2 are two distinct inverses of $a \mod N$. Then,

$$x_1 = x_1 \cdot 1 = x_1 \cdot ax_2 = 1 \cdot x_2 = x_2 (\mod N)$$

which is a contradiction.

1.24. Since p is a prime, all elements in the given range that are not a multiple of p have inverse $\mod p^n$. Among the numbers $\{0, \cdots, p^n - 1\}$ only p^{n-1} are multiples of p. Therefore, $p^n - p^{n-1} = p^{n-1}(p-1)$ numbers have inverses.

1.25. By Fermat's Little Theorem, $2^{126} \equiv 1 \mod 127$. We can write $2^{126} = 2^{125} \cdot 2$. So 2^{125} is the inverse of $2 \mod 127$. By observation (or by extended Euclid's algorithm) we deduce that $2 \cdot 64 = 128 \equiv 1 \mod 127$, so $2^{125} \equiv 64 \mod 127$ since the inverse is unique.

1.26. Take $p = 2$, $q = 5$ so that $pq = 10$. First observe that $17^{(2-1)(5-1)} = 17^4 = 1 \mod 10$. Hence,

$$17^{17^{17}} \mod 10 = 17^{(17^{17} \mod 4)} \mod 10$$

Also, $17 = 1 \mod 4$. So, $17^{17} \mod 4 = 1^{17} \mod 4 = 1$ This gives

$$17^{17^{17}} \mod 10 = 17 \mod 10 = 7 \mod 10$$

so the least significant decimal digit is 7.

1.27. First calculate $(p-1)(q-1) = 16 \cdot 22 = 352$. We use the extended Euclid algorithm to compute the $\gcd(3, 352)$ and get the inverse d of $e \mod 352$. We easily obtain $e \cdot d \equiv 1 \mod 352 \Rightarrow d \equiv -117 \equiv 235 \mod 352$.
The encryption of the message $M = 41$ is $E(M) = M^e \mod N = 41^3 = 117 \cdot 41 = 105 \mod 391$.

1.28. We first calculate $(p-1)(q-1)$ which in our case is $6 \cdot 10 = 60$. Then we need to come up with an e which is relatively prime to 60 so that it has an inverse d. We observe that $e = 11$ has $\gcd(11, 60) = 1$ and $11 \cdot 11 = 1 \mod 60$, therefore the values $e = 11$ and $d = 11$ are appropriate. Other good pairs are $(7, 43), (13, 37), (17, 53), (19, 59), (23, 47), (29, 29), (31, 31), (41, 41))$.

1.29. (a) Here H is the same as in the example of pg.46 of the book, only with 2 coefficients instead of 4. With the same reasoning as the proof of the Property in pg.46, we assume that $X_2 \neq y_2$ and we want to determine the probability that equation $a_1(x_1 - y_1) = a_2(y_2 - x_2)$ holds. Assuming we already picked a_1, that probability is equal to $1/m$, since the only way for the equation to hold is to pick a_2 to be $(y_2 - x_2)^{-1} \cdot a_1 \cdot (x_1 - y_1) \mod m$. We can see that, since m is prime, $(y_2 - x_2)^{-1}$ is unique. Thus H is universal. We need $2 \cdot \lceil \log m \rceil$ bits.
(b)H is not universal, since according to (a) we need a unique inverse of $(y_2 - x_2) \mod m$. For this to hold m has to be prime, which is not true (unless $k = 1$). We need $2k$ bits.
(c)We calculate $P = \Pr[f(x) = f(y)]$ for $x \neq y$. We have $P = \sum_{i=1}^{m-1} \frac{1}{(m-1)^2} = \frac{1}{m-1}$. Thus H is universal. The total number of functions $f : [m] \to [m-1]$ is $(m-1)^m$ so we need $m \log(m-1)$ bits.

1.30. (a)We assume for simplicity that $n = 2^k$ for some fixed k. We perform addition as follows: first split the numbers into 2^{k-1} distinct pairs and perform addition within each pair, using lookahead circuits. Continue by splitting the result into distinct pairs each time and perform addition within each pair. We construct this way a complete binary tree with $\log n$ levels, each level using $\log m$ extra levels for addition (if n was not a power of 2 the tree would simply not be complete). Total we have $O((\log n)(\log m))$ depth.
(b) We let the i-th bit of r be the result of the addition of the i-th bits of all three numbers (i.e. $0 + 0 + 0 = 0, 0 + 1 + 0 = 1, \cdots, 1 + 1 + 0 = 0, \cdots, 1 + 1 + 1 = 1$). We also let the $i + 1$ bit of s to be the carry of the addition of the i-th bit of the three numbers (i.e. if two or three of the i-th bits of the three numbers are 1, then the carry is 1). It is straightforward that $x + y + z = r + s$.
(c) In multiplication, we have to add together n copies of x appropriately shifted. We add these copies by splitting the numbers into triplets and performing the trick from (b). We repeat for k levels until there are less than 3 numbers left, i.e. when $n \cdot (\frac{2}{3})^k = 1 \Rightarrow k = \frac{\log n}{\log 3 - \log 2} = O(\log n)$.

1.31. (a) By the equation $\log N! = \log 1 + \log 2 + \cdots + \log N$ we can easily see that $N!$ is approximately $\Theta(N \cdot \log N) = \Theta(N \cdot n)$ bits long.
(b)We can compute $N!$ naively as follows:

```
factorial (N)
    f = 1
    for i = 2 to N
        f = f · i
```

Running time : we have N iterations, each one multiplying two $N \cdot n$-bit numbers (at most). By 1.30 the running time is $O(N \cdot \log(N \cdot n) = O(N \cdot n)$.

1.32. (a) Assume that N consists of n bits. We can then perform a binary search on the interval $[2^n - 1, 1]$ to find if it contains a number q such that $q^2 = N$. Every iteration takes time $O(n^2)$ to square the current element and $O(n)$ to compare the result with N. As there are $O(\log 2^n)$ iterations, the total running time is $O(n^3)$.

(b) $N = q^k \Rightarrow \log N = k \log q \Rightarrow k = \frac{\log N}{\log q} \leq \log N$ for all $q > 1$. If $q = 1$, then we must have $N = 1$.

(c)We first give an algorithm to determine if a n-bit number N is of the form q^k for some given k and $q > 1$. For this , we use the same algorithm of part (a) only instead of squaring, we will raise numbers to the k-th power and check if we obtain N. This will take $O(n)$ iterations: moreover, each powering operation takes time at most $\sum_{i=1}^{k}(in \cdot n) = O(k^2 n^2)$. Hence, one run of this algorithm takes time $O(k^2 n^3)$. To check if N is a power, we need to repeat this for all $k \leq \log N \leq n$. This yields a running time of $O(n^6)$.

1.33. The least common multiple (lcm) of any two numbers x, y can easily be seen to equal $\text{lcm}(x, y) = (x \cdot y)/\gcd(x, y)$. We therefore need $O(n^3)$ operations to compute the gcd, $O(n^2)$ operations to multiply x and y and $O(2n \cdot n) = O(n^2)$ operations to divide. Total $O(n^3)$ running time.

1.34. **Solution 1 :**
We calculate the expected (average) value of the number of times we must toss a coin before it comes up heads. Let X be the random variable corresponding to the number of tosses needed before coming up heads.

$$E[X] = \sum_{i=1}^{\infty} i \cdot P[X = i] =$$

We now calculate $P[X = i]$: For a coin to come up heads (for the first time) in exactly i tosses, we must have $i - 1$ tails followed by one head. It follows that $P[X = i] = (1 - p)^{(i-1)}p$, for $i \geq 1$. So,

$$E[X] = \sum_{i=1}^{\infty} i \cdot (1 - p)^{(i-1)}p = p \cdot \frac{d}{dp}(\sum_{i=0}^{\infty} -(1 - p)^i)$$

The above powerseries $\sum_{i=0}^{\infty} -(1 - p)^i$ converges for $0 < p < 1$ and it is equal to $-\frac{1}{1-(1-p)} = -\frac{1}{p}$. After taking the derivative, we obtain

$$E[X] = p \cdot \frac{1}{p^2} = \frac{1}{p}$$

Solution 2 :
Let X be the random variable as in solution 1. Then $E[X]$ the average number of tosses, with each possibility weighted by its probability. With probability probability p we get heads in one toss of the coin and with probability $1 - p$, we get tails and need to start again and do another $E[X]$ on average (hence we do a total of $E[X] + 1$ tosses with probability $1 - p$). Therefore $E = p \cdot 1 + (1 - p) \cdot (1 + E) = 1 + (1 - p)E$.

1.35. (a) For a number $1 \leq n < p$ to be its own inverse modulo p it is necessary to have $n^2 - 1 = k \cdot p$, a multiple of p. This comes from the gcd formula : $\gcd(n, p) = 1 = n^n - k \cdot p$. Equivalently, $n^2 - 1 = (n-1)(n+1) \equiv 0 \mod p$. Solving for n we get $n = +1, p-1$. We observe that for all those values n is indeed its own inverse.

(b) Among the $p-1$ numbers, 1 and $p-1$ are their own inverses and the rest have a (different than themselves) unique inverse $\mod p$. Since inversion is a bijective function, each number a in $\{2, \cdots, p-2\}$ is the inverse of some number in the same range not equal to a. Thus,$(p-2)(p-3)\cdots 2 \equiv 1 \mod p \Rightarrow (p-1)! \equiv (p-1) \equiv -1 \mod p$.

(c) Assume, towards contradiction that N is not a prime and also $(N-1)! \equiv -1 \mod N$. Then $(N-1)! = -1 + kN \Rightarrow 1 = -(N-1)! + kN$ which implies that $\gcd((N-1)!, N) = 1$. This is false if N not a prime since there exists some prime $q < N$ that is a multiple of N which appears in the factorial product above.

(d) This rule involves calculating a factorial product which takes time exponential on the size of the input. Thus, the algorithm would not be efficient.

1.36. (a) $p \equiv 3 \mod 4$ implies that $p = 4k + 3 \Rightarrow p + 1 = 4(k+1) \Rightarrow \frac{(p+1)}{4} = k + 1$, an integer.

(b) By Fermat's Little Theorem, $a^{p-1} = 1 \mod p$, so $(a^{\frac{p-1}{2}} - 1)(a^{\frac{p-1}{2}} + 1) = 0 \mod p$. Suppose $a^{\frac{p-1}{2}} + 1 = 0 \mod p$ and x is the square root of a; then we have $x^{p-1} = -1 \mod p$, contradicting Fermat's Little Theorem. Hence, we must have $a^{\frac{p-1}{2}} - 1 = 0 \mod p$, which means $a^{\frac{p+1}{2}} = a \mod p$. This shows that $\left(a^{\frac{p+1}{4}}\right)^2 = a \mod p$. By a), $\frac{p+1}{4}$ is an integer, so that $a^{\frac{p+1}{4}}$ is well defined.

1.37. (a) Make the table and observe that each number has a different pair (i, j) of residues $\mod 3$ and $\mod 5$.

(b) Assume , towards contradiction that both i and i' with $i \neq i'$ have the same (j, k). Then $i = Ap + j = Bq + k$ and $i' = Cp + j = Dq + k$. So $i - i' = (A - C)p = (B - D)q$. Since p, q are different primes, q has to divide $(A - C)$ and p has to divide $(B - D)$. So $i - i' = 0 \mod pq$. Which means $i = i'$ since both are $< pq$.

Assume now, towards contradiction, that there is a particular pair (j, k) such that there is no integer $i < pq$ with the desired property. The total number of pairs is pq (which is the same as the number of numbers mod pq) so there must be at least two different numbers i, i' that have the same pair (j', k'), contradiction.

(c) . By (b) it is enough to show that the expression in brackets has the property $i = j \mod p$ and $i = k \mod q$.

$$\begin{aligned} i = & \quad \{j \cdot q \cdot (q^{-1} \mod p) + k \cdot p \cdot (p^{-1} \mod q)\} \mod pq \mod p \\ = & \quad \{j \cdot q \cdot (q^{-1} \mod p) + k \cdot p \cdot (p^{-1} \mod q)\} \mod p \mod pq = j \end{aligned}$$

and

$$\begin{aligned} i = & \quad \{j \cdot q \cdot (q^{-1} \mod p) + k \cdot p \cdot (p^{-1} \mod q)\} \mod pq \mod q \\ = & \quad \{j \cdot q \cdot (q^{-1} \mod p) + k \cdot p \cdot (p^{-1} \mod q)\} \mod q \mod pq = k \end{aligned}$$

(d) Part (b) is immediately extended to more than two primes. Part (c) will give the expression for i :

$$\begin{aligned} i = \{j_1 \cdot q_2 \cdot q_3 \cdot \cdots \cdot q_n((q_2 \cdot q_3 \cdot \cdots \cdot q_n)^{-1} \mod q_1) + \\ + \cdots + \\ + j_n \cdot q_1 \cdot q_2 \cdot \cdots \cdot q_{n-1}((q_1 \cdot q_2 \cdot \cdots \cdot q_{n-1})^{-1} \mod q_n) \mod p_1 p_2 \cdots p_n \end{aligned}$$

1.38. We will show (a) and (b) together. For (b) observe that is we break a decimal number N into r-tuples, say, $N_k N_{k-1} \cdots N_1$ then $N = N_1 + N_2 \cdot 10^r + \cdots + N_k \cdot (10^r)^{k-1}$. Therefore, for our divisibility criterion, we try to achieve the following :$N \mod p = N_1 + N_2 \cdot 10^r + \cdots + N_k \cdot (10^r)^{k-1} = N_1 + N_2 + \cdots + N_k$. For this equation to hold, we need $10^r = 1 \mod p$. From Fermat's Little Theorem, we know that $10^{p-1} = 1 \mod p$, so looking for the smallest such r we will get a divisor of $p-1$. For $p = 13$ we can check that the choice $r = 13 - 1 = 12$ is the smallest choice for r, while for $p = 17$ the smallest choice is $r = 17 - 1 = 16$.

1.39. From Fermat's Little Theorem, we know that $a^{p-1} = 1 \mod p$. Therefore,

$$a^{b^c} = a^{b^c \mod (p-1)} \mod p$$

we first need to calculate $b^c \mod (p-1)$. Note that this is modular exponentiation and is hence doable in polynomial time. We repeatedly square b modulo $(p-1)$ and compute $b^c \mod (p-1)$ in the same way as computing the n^{th} power of a matrix. Since we do $O(n)$ multiplications, each taking $O(n^2)$ time, the total time is $O(n^3)$. Let $b^c \mod (p-1) = d$. Then, by the equality $a^{b^c} \mod p = a^d \mod p$, all we need to do is use again the repeated squaring algorithm. So the total running time of our algorithm is $O(n^3)$.

1.40. Assume, towards contradiction that N is a prime and still x is a non-trivial square root of $1 \mod N$. Then $x^2 = 1 + kN \Rightarrow (x-1)(x+1) = kN$. Since N is a prime it must divide at least one of $(x-1)$,$(x+1)$ which leads to conclude $x \equiv 1$ or $-1 \mod N$, contradiction.

1.41. (a) Assume $a \equiv y^2 \mod N$ for $1 \leq y < N$. Then $x^2 - y^2 \equiv 0 \mod N \Rightarrow (x-y)(x+y) \equiv 0 \mod N$. This means either $x - y \equiv 0 \mod N \Rightarrow x = y$ since both $x, y < N$ or $x + y \equiv 0 \mod N \Rightarrow x = N - y$ since $x, y < N$. These are the only two possible values.
(b)We will first show that for any two numbers x, y in the range $\{1, 2, \cdots, \frac{N-1}{2}\}$ we have $x^2 \neq y^2 \mod N$. Assume towards contradiction that $x^2 \equiv y^2 \Rightarrow (x-y)(x+y) \equiv 0$. Since N is prime, $x \neq y$ this means $x + y \equiv N$. But both $x, y \leq \frac{N-1}{2}$ so $x + y \leq N - 1$, contradiction.
Therefore, each of those $\frac{N-1}{2}$ numbers defines a quadratic residue $\mod N$. Also, from (a), the rest of the $\frac{N-1}{2}$ numbers in the range $\{\frac{N-1}{2} + 1, \cdots N-1\}$ lead to the same residues. Adding 0 as a quadratic residue, we have exactly $\frac{N+1}{2}$ of them.
(c)Take, say, $N = 15$ and $a = 1$ then the equation $x^2 \equiv 1 \mod 15$ has solutions $1, -1, 4$.

1.42. We just need to calculate the inverse of $e \mod (p-1)$. But this we can do by using Extended Euclid's algorithm for the $\gcd(e, p-1) = 1$.

1.43. First pick an appropriate message m such that $m^{\frac{ed-1}{2}}$ is neither 1 nor $-1 \mod N$. Then $m^{ed-1} \equiv 1 \mod N \Rightarrow (m^{\frac{ed-1}{2}} - 1)(m^{\frac{ed-1}{2}} + 1) \equiv 0 \mod N \Rightarrow$ one of the $\gcd(N, (m^{\frac{ed-1}{2}} - 1))$ or $\gcd(N, (m^{\frac{ed-1}{2}} + 1))$ is a non-trivial factor of N. Since $N = qp$ and we have determined, say p, we can just divide $N/p = q$.

1.44. The three messages that Alice sends are M^3
$mod N_1$, $M^3 \mod N_2$, $M^3 \mod N_3$, where $M < min\{N_1, N_2, N_3\}$. By the Chinese remainder theorem there is a unique number $x < N_1 \cdot N_2 \cdot N_3$ with $x = M^3 \mod N_1$,$x = M^3 \mod N_2$,$x = M^3 \mod N_3$. We observe that $M^3 < N_1 \cdot N_2 \cdot N_3$ and also has the above residues. Therefore, we can just compute M^3 by the equations in 1.37d) and then take the cubic root i $\mathbb{Z}$ (recall we are not able to take roots in Z_N, but this is not necessary here).

1.45. a) The digital signature scheme can be used to both authenticate the identity of the sender of the message and to ensure that the message has not been altered by a third party during communication.

b) `verify`$((N, e), M^d, M)$ can be implemented by checking whether $(M^d)^e \mod N$ equals M. Then, if the signature was created by the private key d, we have $(M^d)^e \mod N = (M^e)^d \mod N = M$

mod $N = M$, by the correctness of the RSA protocol. Conversely, if an adversary was able to sign given only (N, e), the adversary would be able to exponentiate by $d \mod N$, which would allow him to decrypt, contradicting the security of the RSA protocol. Hence, if the RSA is secure, so is this scheme for digital signatures.

c) I picked $p = 137, q = 71$. Hence $N = pq = 9727$ and $\Phi(N) = (p-1)(q-1) = 9520 = 2^4 \cdot 5 \cdot 7 \cdot 17$. Then, I chose $e = 99$, which is coprime to $\Phi(n)$. d must then be the inverse of $e \mod \Phi(n)$, that is 6539, found by running Extended Euclid. The first letter of my name is L and I choose it to map to binary using the $ASCII$ code, for which $L = 76$. Then, the signature of this first number is $76^{99} \mod 9727 = 4814$. Finally, $4814^{99} \mod 9727 = 76$, as required.

d) 391 can be factored as $17 \cdot 23$. Then, $\Phi(391) = 16 \cdot 22 = 352$. Then $d = (e)^{-1} \mod 352 = 145$, by Extended Euclid. In fact, $145 * 17 = 2465 = 1 \mod 352$, as required.

1.46. a) When Eve intercepts the encrypted message $M^e \mod N$ sent by Alice to Bob, she can just ask Bob to sign it for her with his private key, to obtain $(M^e)^d \mod N = M$, by the correctness of the RSA.

b) In this case, Eve can pick k coprime to N at random and ask Bob to sign $M^e \cdot k^e \mod N$. This will yield $(Mk)^{ed} \mod N = Mk \mod N$. Then Eve can use Extended Euclid to obtain $k^{-1} \mod N$ and multiply by such inverse to find M. Notice that in this way Bob's signatures are distributed uniformly over all numbers invertible $\mod N$, as Mk is equally likely to be any of such numbers.