1. Exercise 2.30. Show complete details for each argument, clearly indicating which axioms or previously-proved statements you are using.

Solution. Each of these can be proved from the axioms in various different ways. What follows below is by no means the only method.

(a) Suppose that 0 and 0′ are both identity elements for +. Then it follows that 0 + 0′ = 0′ (since 0 + a = a for all a ∈ R), and also 0 + 0′ = 0 (since a + 0′ = a for all a ∈ R). Therefore in fact 0 = 0′. So there cannot be two different additive identities.

(b) Suppose that 1 and 1′ are both identity elements for ∗. Then 1 ∗ 1′ must be equal to both 1′ and 1, since a ∗ 1′ = a and 1 ∗ a = a for all a ∈ R. So 1 = 1′; there cannot be two different multiplicative identities.

(c) Suppose a ∈ R, and that b, b′ are both additive inverses for a. We wish to show that b = b′. This will imply that there cannot be two different additive inverses for a. Consider the expression (b + a) + b′. Since b is an additive inverse for a, this is equal to 0 + b′. Since 0 is the additive identity, this in turn is equal to b′. On the other hand, by associativity the original expression is equal to b + (a + b′). Since b′ is an additive inverse, this is equal to b + 0, and hence equal to b. So the original expression is equal to both b and b′, and therefore b = b′.

(d) Since ∗ is commutative, we know already that a ∗ 0 = 0 ∗ a. It is necessary only to prove that a ∗ 0 = 0. Consider the expression a ∗ 0 + a ∗ 1. By distributivity, this is equal to a ∗ (0 + 1). Since 0 is an additive identity, this is equal to a ∗ 1. So a ∗ 0 + a ∗ 1 = a ∗ 1. Equivalently, a ∗ 0 + a = a (since 1 is the multiplicative identity). We know that a has an additive identity. Call it −a. Then a ∗ 0 + a + (−a) = a + (−a), and on both sides a + (−a) may be replaced by 0. Therefore a ∗ 0 + 0 = 0, hence a ∗ 0 = 0.

(e) By definition of additive inverse, (−a) + (−a) = 0. On the other hand, we know that (−a) + a = 0, by definition of −a. So both −(−a) and a are additive inverses of −a. It follows from part (c) that they are the same.

(f) Consider the expression (1 + (−1)) ∗ (−1). On the one hand, we know that 1 + (−1) = 0, and (from part (d)) 0 ∗ (−1) = 0. Therefore this expression is equal to 0. On the other hand, we may apply the distributive property to see that this expression is equal to 1 ∗ (−1) + (−1) ∗ (−1). Since 1 is the multiplicative identity, this is equal to (−1) + (−1) ∗ (−1). So we have obtained the equation 0 = (−1) + (−1) ∗ (−1). So (−1) ∗ (−1) is the additive inverse of −1. Since (−1) + 1 = 0, this inverse must be equal to 1.

(g) By part (d), b ∗ 0 = 0. By definition of divisibility, this means that b ∣ 0.

(h) Suppose that a ∈ R has multiplicative inverses b and c. Then b ∗ a = 1 and also a ∗ c = 1. Therefore the expression (b ∗ a) ∗ c (which is equal to b ∗ (a ∗ c) by associativity) is equal to both 1 ∗ c = c and also to b ∗ 1 = b. So in fact b = c. So a cannot have two different multiplicative inverses.

2. Exercise 2.32.

Solution. By definition of congruence, there exist elements k, h ∈ R such that a1 − a2 = k ∗ m and b1 − b2 = h ∗ m, i.e. a1 = a2 + k ∗ m and b1 = b2 + h ∗ m.
Therefore $a_1 \pm b_1 = (a_2 + k \cdot m) \pm (b_2 + h \cdot m)$. By associativity and commutativity, this is equal to $a_2 \pm b_2 + k \cdot m \pm h \cdot m$. By distributivity, this is $a_2 \pm b_2 + (k \pm h) \cdot m$. Hence $(a_1 \pm b_2) - (a_2 \pm b_2) = (k \pm h) \cdot m$, which shows that $(a_1 \pm b_1) - (a_2 \pm b_2)$ is divisible by $m$, i.e. $a_1 \pm b_1 \equiv a_2 \pm b_2$ (mod $m$).

Next, note that by distributivity, $a_1 \cdot b_1 = (a_2 + k \cdot m) \cdot (b_2 + h \cdot m) = a_2 \cdot b_2 + a_2 \cdot h \cdot m + k \cdot m \cdot b_2 + k \cdot m \cdot h \cdot m$, so $a_1 \cdot b_1 - a_2 \cdot b_2 = (a_2 \cdot h + k \cdot b_2 + k \cdot h \cdot m) \cdot m$, which is divisible by $m$. So $a_1 \cdot b_1 \equiv a_2 \cdot b_2$ (mod $m$).

3. Exercise 2.33.

**Solution.** Observe first of all that two congruence classes $\overline{a}$ and $\overline{b}$ are equal if and only if $a_1 \equiv a_2$ (mod $m$). It is fine if you did not prove this explicitly, but here is a proof for completeness. First, if $\overline{a} = \overline{b}$, then certainly $a_2 \in \overline{a}$, hence $a_1 \equiv a_2$ (mod $m$). For the converse, suppose that $a_1 \equiv a_2$ (mod $m$). Then it follows that anything congruent to $a_1$ is also congruent to $a_2$ and vice versa, since one can verify that $\equiv$ is a symmetric and transitive relation. So $\overline{a} = \overline{b}$.

We first must check that addition is well-defined in $R/(m)$. This amounts to checking that there is no ambiguity in writing $\overline{a} + \overline{b} = a + b$. The only way ambiguity can arise is if the congruence class $a + b$ depends not only on the congruence classes $a$ and $b$, but on the specific choice of elements $a, b$ from them. So we must verify that if $a_1, a_2$ are elements of the congruence class $\overline{a}$, and $b_1, b_2$ are elements of the congruence class $\overline{b}$, then $a_1 + b_1 = a_2 + b_2$. But this amounts to saying that if $a_1 \equiv a_2$ (mod $m$) and $b_1 \equiv b_2$ (mod $m$), then $a_1 + b_1 \equiv a_2 + b_2$ (mod $m$). This was verified in the previous exercise. So the expression $a + b$ is unambiguous: its value is the same regardless of the choice of $a$ and $b$ from their respective congruence classes.

Similarly, we must check that multiplication is well-defined, which proceeds in an exactly analogous way: with the same notation as before, we have that $a_1 \cdot b_1 \equiv a_2 \cdot b_2$ (mod $m$) (by the previous exercise), hence $a_1 \cdot b_1 = a_2 \cdot b_2$. So the definition $\overline{a} \cdot \overline{b} = a \cdot b$ is unambiguous.

Next we must verify that the set $R/(m)$, with operations $+, \cdot$, satisfies all of the axioms of a ring. All of these follow readily from the fact mentioned in the first paragraph, but we treat them in turn for completeness.

- **Identity law for $+$:** observe that for all $\overline{a} \in R/(m)$, $\overline{a} + \overline{0} = \overline{a + 0} = \overline{a}$, and $\overline{0} + \overline{a} = \overline{0 + a} = \overline{a}$. So $\overline{0}$ is an additive identity element.
- **Inverse law for $+$:** For any $\overline{a} \in R/(m)$, observe that $\overline{a} + \overline{-a} = \overline{a + (-a)} = \overline{0}$, the additive identity for $R/(m)$. Similarly, $\overline{-a} + \overline{a} = \overline{0}$. So each element has an additive inverse.
- **Associative law for $+$:** Note that $\overline{a + (b + c)} = \overline{a + b + c} = \overline{(a + b) + c} = \overline{a + b} + \overline{c} = (\overline{a} + \overline{b}) + \overline{c}$, where we have used in the middle that $+$ is associative for $R$ itself.
- **Commutative law for $+$:** Note that $\overline{a + b} = \overline{a + b} = \overline{b + a} = \overline{b + a}$, where in the middle we have used that $+$ is commutative in $R$.
- **Identity, associative, and commutative laws for $\cdot$:** the same proof works as for $+$; simply replace $+$ by $\cdot$ in all places and replace $0$ by $1$.
- **Distributive law:** $\overline{a \cdot (b + c)} = \overline{a \cdot b + a \cdot c} = a \cdot (b + c) = a \cdot b + a \cdot c = \overline{a \cdot b + a \cdot c}$.
4. Exercise 2.35, part (d) only.

**Solution.**

We begin from \(a, b\) and compute successive remainders as follows. At each step, I reduce all coefficient modulo 7.

\[
a \equiv x^5 + 3x^4 + 2x^3 + 4x^2 + 2x + 2 \pmod{7}
\]

\[
b \equiv x^5 + x^4 + 5x^3 + 4x^2 + x + 5 \pmod{7}
\]

\[
r_2 \equiv a - b
\]

\[
r_3 \equiv b - (4x + 3)r_2
\]

\[
r_4 \equiv r_2 - (3x^3 + 6x^2 + 0x + 5)r_3
\]

So 4 is a linear combination of \(a\) and \(b\). By multiplying by the inverse of 4 modulo 7, so is 1. So the greatest common divisor of \(a\) and \(b\) is 1.

5. Exercise 2.36, part (d) only.

**Solution.**

We obtain each \(r_i\) as a linear combination of \(a\) and \(b\) as follows.

\[
r_2 \equiv a - b
\]

\[
r_3 \equiv b - (4x + 3)r_2
\]

\[
r_4 \equiv r_2 - (3x^3 + 6x^2 + 0x + 5)r_3
\]

Since \(r_4 \equiv 4 \pmod{7}\), we can obtain 1 as a linear combination by multiplying through by 2 \(\pmod{7}\), to obtain:

\[
1 \equiv (3x^4 + 3x^3 + x^2 + 5x + 4)a + (4x^4 + 5x^3 + x^2 + 2x)b.
\]
Problem Set 10

6. Consider the following variant of EC Diffie-Hellman key exchange, in which Alice and Bob only exchange individual numbers, rather than both coordinates of a point on an elliptic curve.

- **Public parameter creation:** same as in table 6.5.
- **Private computations:** same as in table 6.5.
- **Public exchange of values:** Alice sends the $x$-coordinate of $Q_A$ to Bob; Bob sends the $x$-coordinate of $Q_B$ to Alice.
- **Further private computations:** Both Alice and Bob determine the $x$-coordinate of $(n_A \cdot n_B)P$. This is their shared secret value.

(a) Prove that if $Q, Q'$ are two points on an elliptic curve with the same $x$-coordinate, and $n$ is any integer, then $nQ$ and $nQ'$ also have the same $x$-coordinate.

(b) Describe how Alice is able to (efficiently) determine the shared secret, using only the information that she knows. You may assume that Alice has an efficient algorithm to determine square roots modulo $p$.

(c) What advantages, if any, does this system have over the usual ECDH system described in table 6.5?

**Note.** You will implement a function to perform the task Alice and Bob must perform in the last programming problem.

**Solution.**

(a) If $Q = (x, y)$ and $Q' = (x, y')$ are two points with the same $x$-coordinate, then $y^2 \equiv y'^2 \pmod{p}$, hence $(y + y')(y - y') \equiv 0 \pmod{p}$, and either $y' \equiv y \pmod{p}$ or $y' \equiv -y \pmod{p}$. In the first case, $Q = Q'$, so also $nQ = nQ'$ and these certainly have the same $x$-coordinate. In the second case, we have $Q = \ominus Q$, which is the same thing as $(-1)Q$. Therefore in this case $nQ' = (-n)Q = \ominus nQ$. Hence $nQ$ and $nQ'$ are inverses, which means that they have the same $x$-coordinate and inverse $y$-coordinates.

(b) Suppose that Alice receives the coordinate $x_B$ from Bob (the $x$-coordinate of $Q_B$). Then she can compute $z \equiv x_B^3 + Ax_B + C \pmod{p}$. The $y$-coordinate of $Q_B$ is a square root modulo $p$ of $z$. Using her square root algorithm, Alice can therefore extract one of the two square roots of $z$ modulo $p$; suppose that she computes the number $y_B'$. Then $Q'_B = (x_B, y'_B)$ is a point on the elliptic curve, with the same $x$-coordinate as $Q_B = n_BP$. Alice may now compute $n_A \cdot Q'_B$, and extract its $x$-coordinate. By part (a), the $x$-coordinate of this point is the same as the $x$-coordinate of $n_A Q_B = (n_A \cdot n_B)P$. Therefore Alice has successfully extracted the intended shared secret.

(c) This system involves less transmitted information (by a factor of 2) than ordinary elliptic curve Diffie-Hellman, since only one coordinate must be sent from Alice to Bob and vice versa. The cost of this decreased bandwidth requirement is an increased computation requirement: Alice and Bob must both extract a square root modulo $p$. The resulting shared secret contains one fewer bit of information, although this is unlikely to be significant.

Note that essentially the same trade-offs arise if Alice and Bob use ordinary elliptic curve Diffie-Hellman coupled with point compression (as discussed in exercise 6.18 and at the end of section 6.4.2): in that case, Alice and Bob still transmit enough information to recover the point $n_A n_B P$ completely by transmitting one more bit each, at the cost of having to do the computation involved in the modular square root.

Solution.

(a)

\[ h \equiv f^{-1}g \pmod{q} \]
\[ \equiv 19928^{-1} \cdot 18643 \pmod{918293817} \]
\[ \equiv 767748560 \]

(b) Following the steps on page 375:

\[ a = f \cdot e \pmod{q} \]
\[ = 19928 \cdot 619168806 \pmod{918293817} \]
\[ = 600240756 \]
\[ b = a \cdot f^{-1} \pmod{g} \]
\[ = 600240756 \cdot 9764 \pmod{18643} \]
\[ = 11818 \]

So the plaintext is 11818.

(c) Bob sends \((rh + m) \pmod{q} = (19564 \cdot 767748560) \pmod{918293817} = 619167208\).

Programming problems

Full formulation and submission: [https://www.hackerrank.com/m158-2016-pset-10](https://www.hackerrank.com/m158-2016-pset-10)

8. Implement functions to add and multiply elements of the polynomial ring \( \mathbb{F}_p[x] \).

Solution.

Here is a sample implementation of functions to add and multiply polynomials. Polynomials are representing as lists of coefficients, which are assumed to end with a nonzero coefficient.

```python
def FPX_add(a,b,p):
    d = len(a)-1
    e = len(b)-1
    c = [0]*(max(d,e)+1)
    for i in xrange(d+1):
        c[i] += a[i]
    for i in xrange(e+1):
        c[i] += b[i]
    for i in xrange(max(d,e)+1):
        c[i] %= p
    return trim(c)

def FPX_mult(a,b,p):
    d = len(a)-1
    e = len(b)-1
    c = [0]*(d+e+1)
```

Due the night of Monday 12/5 (hard deadline 4am on 12/6).
for i in xrange(d+1):
    for j in xrange(e+1):
        c[i+j] += a[i]*b[j]
for i in xrange(d+e+1):
    c[i] %= p
return trim(c)

# Given a list of integers, this function trims off any zeros from the end.
def trim(a):
    trimmedLen = len(a)
    while trimmedLen > 0 and a[trimmedLen-1] == 0:
        trimmedLen -= 1
    return a[:trimmedLen]

9. Implement a function to perform division with remainder in $\mathbb{F}_p[x]$. That is, given two polynomials $a, b \in \mathbb{F}_p[x]$, you must determine polynomials $k, r$ (the quotient and the remainder) such that $a = k \cdot b + r$ and $\deg r < \deg b$.

Solution.

Here is a sample implementation, roughly following the notation and method of Proposition 2.44 in the textbook.

```python
def FPX_divrem(a,b,p):
    k = [] # Zero polynomial, as a list
    r = a[:]
    while len(r) >= len(b):
        d = len(b)-1
        e = len(r)-1
        coeff = r[e]*modinv(b[d],p) % p
        k = FPX_add(k, [0]*(e-d) + [coeff], p)
        rmod = FPX_mult(b, [0]*(e-d) + [-coeff], p)
        r = FPX_add(r,rmod,p)
    return k,r
```

10. Implement the extended Euclidean algorithm for $\mathbb{F}_p[x]$. You will be given polynomials $a, b$ of different degree$^1$ and must find polynomials $u, v, g$ such that $au + bv = g$, and $g$ is the greatest common divisor of $a$ and $b$. You should return a choice of $u, v, g$ such that $g$ is monic (i.e. the highest-degree coefficient is 1) and $u, v$ have the smallest possible degree.

Solution.

Here is an implementation, following roughly the same strategy as the usual Euclidean algorithm.

---

$^1$It is not essential to assume that the degrees are different, but it may simplify some analysis.
Problem Set 10

# Computes and returns a-b*k in FPX
# Done often enough that it’s useful to bubble it off
def FPX_minusmult(a, b, k, p):
    diff = FPX_mult(b, k, p)
    ndiff = FPX_mult([-1], diff, p)
    return FPX_add(a, ndiff, p)

def FPX_euclid(a, b, p):
    u0, v0, g0 = [1], [0], a
    u1, v1, g1 = [0], [1], b
    while len(g1) > 0:
        q, r = FPX_divrem(g0, g1, p)
        u2 = FPX_minusmult(u0, u1, q, p)
        v2 = FPX_minusmult(v0, v1, q, p)
        g2 = r
        u0, v0, g0 = u1, v1, g1
        u1, v1, g1 = u2, v2, g2
    # Turn g0 into a monic polynomial
    coeff = g0[-1]
    cinv = modinv(coeff, p)
    u = FPX_mult(u0, [cinv], p)
    v = FPX_mult(v0, [cinv], p)
    g = FPX_mult(g0, [cinv], p)
    return u, v, g

11. Given an elliptic curve over $\mathbf{F}_p$, where $p \equiv 3 \pmod{4}$, a positive integer $n$, and \textit{the x-coordinate of a point} $P$ on the curve, determine the \textit{x-coordinate of $nP$}. Such a function could be used to do the variant of EC Diffie-Hellman mentioned in problem 6. You will want to use a fact mentioned in Proposition 2.26 in order to do part of this computation.

\textbf{Solution.}

We can simply “decompress” the point by finding a square root of $x^3 + Ax + B \pmod{p}$, using it as the $y$-coordinate, and multiplying the resulting point by $n$.

Using Proposition 2.26, we can find this square root simply by evaluating $(x^3 + Ax + B)^{(p+1)/4} \pmod{p}$.

### Omitted: function mult(P,n,A,B,p) to do elliptic curve multiplication mod p
### (and helper functions)

# Modular square root, assuming that $p = 3 \pmod{4}$.
def sqroot(a, p):
    assert p%4 == 3
    return pow(a, (p+1)/4, p)

def decompress(x, A, B, p):
    y = sqroot((x**3 + A*x + B)\%p, p)
    return (x, y)
def pcMult(x,n,A,B,p):
    P = decompress(x,A,B,p)
    Q = mult(P,n,A,B,p)
    assert Q != 0
    return Q[0]

# I/O

A,B,p = map(int,raw_input().split())
x = int(raw_input())
n = int(raw_input())
print pcMult(x,n,A,B,p)

12. (Extra credit) Do the same task as the previous problem, in the case \( p \equiv 1 \pmod{4} \). You will need to do some research to find an efficient algorithm to compute square roots modulo \( p \).

Solution.

It is necessary to implement a modular square root function that will work for primes congruent to 1 \( \pmod{4} \). One algorithm to perform this task is the Tonelli-Shanks algorithm, which you can read about in various places. Here is an implementation of this algorithm. Replacing the \( \text{sqroot} \) function from the previous solution with this one gives a solution to the more general problem.

### Omitted: function \( \text{inverse}(a,m) \) for inverses modulo \( m \).

def sqroot(a,p):
    if a%p == 0: return 0

    s = p - 1
    k = 0
    while s % 2 == 0:
        s /= 2
        k += 1

    # Write \( a = b \ u^2 \), where \( b \)'s order is a power of 2.
    # This reduces the problem to the square root of \( b \).
    u = pow(a,(s+1)/2,p)
    b = inverse(pow(a,s,p),p)

    g = 2
    while pow(g,(p-1)/2,p) == 1:
        g += 1
    g = pow(g,s,p) # g is an order \( 2^k \) element

    # Loop invariants:
    # \( a = b \ u^2 \pmod{p} \)
    # \( \text{ord} g = 2^k \)
# $b^{2^{(k-1)}} = 1 \mod p$

while b != 1:
    assert k >= 1
    if pow(b, 1 << (k-2), p) == p-1:
        u = u*g % p
        b = b*inverse(g*g, p) % p
        g = g * g % p
        k -= 1

    assert u*u % p == a
return u