Problem Set 8 Solutions

Problem 8-1. Modular Operations

For this problem, do not assume that arithmetic operations have $O(1)$ cost. You may assume that the operations $(a \cdot b)$ and $(a \mod b)$ may be done in time $O(\log a \log b)$. On inputs of length $n$, we define an algorithm to be polynomial-time if it runs in $O(n^k)$ time for some constant $k$.

(a) Given $a, b, c$ and prime $p$, give a polynomial-time algorithm that computes $a^{bc} \mod p$. Note that the size of the input is $\log a + \log b + \log c + \log p$.

Solution: By Fermat’s little theorem, $(a^{p-1} \mod p) = 1$. Therefore:

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

1. First compute $(b \mod (p - 1))$ in time $O(\log b \log p)$.
2. Now compute $(b^c \mod (p - 1))$ using repeated squaring. This will consist of $O(\log c)$ multiplications of numbers of size at most $p$. So, each multiplication will take $O(\log^2 p)$ time, for a total of $O(\log c \log^2 p)$ time.
3. Compute $(a \mod p)$ in time $O(\log a \log p)$.
4. Note that $(b^c \mod (p - 1)) < p$. Compute $(a^{b^c} \mod p)$ using repeated squaring in time $O(\log p \log^2 p) = O(\log^3 p)$.

The total running time is $O(\log b \log p + \log c \log^2 p + \log a \log p + \log^3 p)$. This is upper bounded by $O(n^3)$, where $n$ is the size of the input.

(b) Given two integers, $a$ and $b$, give a polynomial-time algorithm to find the closest integer to $\sqrt{a}$.

Solution: The closest integer to $\sqrt{a}$ is an integer less than $a$. So we can search in the set $\{1, 2, \ldots, a - 1\}$ to find it. In order to do this efficiently we use binary search. The following procedure checks if $a$ has a $b$-root in the integer interval $[x..y]$

FIND-ROOT-IN-INTERVAL($a, b, x, y$)

1. Start with $z = (y - x)/2$ and check if $z^b = a$ if so return $z$ else continue.
2. If $z^b < a$ but $(z + 1)^b > a$ then return the closest between $z$ and $z + 1$ else continue
3. If $z^b < a$ call FIND-ROOT-IN-INTERVAL($a, b, z, y$)
   else call FIND-ROOT-IN-INTERVAL($a, b, x, z$)
The running time of the above procedure is $O(\log(y - x) \log(b(\log a)^2))$. This is because we try only $\log(y - x)$ possible candidates and for each of them we perform $\log b$ multiplications (to raise them to the $b$-power using repeated squaring). Each multiplication takes $O((\log a)^2)$ time because all numbers are less than $a$.

To find the $b$-root of $a$ we have to call the procedure FIND-ROOT($a, b$) = FIND-ROOT-IN-INTERVAL($a, b, 1, a$) which takes $O((\log a)^3 \log b)$ time.

(c) Given an integer $x$, give a polynomial-time algorithm to determine if $x$ is a power, i.e. if there exists integers, $a, b \neq 1$ such that $x$ can be written as $x = a^b$. Hint: Use part (b).

Solution:
Notice that the algorithm in part (b) can be modified so that line 1 outputs not only $z$ but even the statement “$a$ is a pure power”.

If $x$ is a pure power (i.e. can be written as $x = a^b$) then $b = \log_a x \leq \log x$. So it is enough to run the procedure FIND-ROOT($x, b$) for each possible $b \leq \log x$.

The running time is $O((\log x)^5)$.

Problem 8-2. Chinese Remainder Theorem

Given integers $p, q, n = pq$, where $p$ and $q$ are prime, there exists a 1-1 and onto mapping between $\mathbb{Z}^*_{n}$ and $(\mathbb{Z}^*_{p}, \mathbb{Z}^*_{q})$, which is quite useful. Let’s explore it. The mapping is $f(x) = (x \mod p, x \mod q)$. You may assume that the operations $(a \cdot b)$ and $(a \mod b)$ may be done in time $O(\log a \log b)$.

(a) Give an algorithm to compute $x$ given $f(x), p, q$.

Solution: Given input $(r, s)$, use the Euclidian algorithm to find $y = q^{-1} \mod p$ and $z = p^{-1} \mod q$. Output $r y + s p z \mod n$. Note that $r y + s p z \equiv r y \equiv r \mod p$ and $r y + s p z \equiv s p z \equiv s \mod p$. Since part (a) is 1-to-1, this mapping is necessarily 1-to-1.

(b) Now, let us define the following multiplication operator: $(r, s) \odot (t, u) = (r t \mod p, s u \mod q)$. Show that it is closed under $(\mathbb{Z}^*_{p}, \mathbb{Z}^*_{q})$.

Solution: Since multiplication is closed over $\mathbb{Z}^*_{p}, rt \mod p \in \mathbb{Z}^*_{p}$. Similarly, $su \mod q \in \mathbb{Z}^*_{q}$.

(c) Ben Bitdiddle designs a new multiplication unit, called the B-Diddy, that multiplies two numbers $x, y \in \mathbb{Z}^*_n$ using the following algorithm:
1. Map $x$ and $y$ into pairs $(r, s), (t, u) \in (\mathbb{Z}^*_p, \mathbb{Z}^*_q)$.
2. Compute $(r, s) \odot (t, u) = (v, w)$.
3. Map \((v, w)\) back into \(z \in \mathbb{Z}_n^*\).
Analyze the B-Diddy’s runtime and compare it to standard multiplication over \(\mathbb{Z}_n^*\).
Which is better to use?

**Solution:**
Multiplying two numbers in \(\mathbb{Z}_n^*\) takes time \(O((\log n)^2) = O((\log p + \log q)^2)\).
Step 1: \(O(\log n \log p + \log n \log q) = O(\log^2 n)\).
Step 2: \(O(\log^2 p + \log^2 q) = O(\log^2 n)\)
Step 3: Running the Euclidian algorithm takes \(O(\log n \log p + \log n \log q) = O(\log^2 n)\). Computing \(r_q y\) and \(s_p z\) take time \(O(\log n \log p + \log n \log q) = O(\log^2 n)\). Assuming w.l.o.g. that \(p > q\), adding \(r_q y\) and \(s_p z\) produces a number of size \(\log n + \log p\) and takes time \(O(\log n + \log p)\). Taking the modulo of this value takes time \(O(\log^2 n)\).
Thus, the B-Diddy takes the same order of complexity as standard multiplication.

(d) Suppose you are given inputs \(p, q, n\) and \((x^3 \mod p, x^3 \mod q)\). Give an algorithm to compute \(x \mod n\). You may assume that \(gcd(3, \phi(n)) = 1\).

**Solution:**
Perform RSA decryption: Compute \(d\) such that \(3d \equiv 1 \mod \phi(n)\). Find \(x^3 \mod n\) using part (b). Compute \(x^{3d} = x^{k\phi(n)+1} = x \mod n\).

Problem 8-3. Snowball Throwing

Several 6,046 students hold a team snowball throwing contest. Each student throws a snowball with a distance in the range from 0 to \(10n\). Let \(M\) be the set of distances thrown by males and \(F\) be the set of distances thrown by females. You may assume that the distance thrown by each student is unique and is an integer. Define a team score to be the combination of one male and one female throw.

Give an \(O(n \log n)\) algorithm to determine every possible team score, as well as how many teams could achieve a particular score. This multi-set of values is called a *cartesian sum* and is defined as:

\[
C = \{m + f : m \in M \text{ and } f \in F\}
\]

**Solution:**
Represent \(M\) and \(F\) as polynomials of degree \(10n\) as follows:

\[
M(x) = x^{a_1} + x^{a_2} + \ldots + x^{a_n}, F(x) = x^{b_1} + x^{b_2} + \ldots + x^{b_n}
\]

Multiply \(M\) and \(F\) in time \(\Theta(n \log n)\) to obtain a coefficient representation \(c_0, c_1, \ldots, c_{2n}\). In other words \(C(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_{2n} x^{2n}\). Each pair \(a_i, b_j\) will account for one term \(x^{a_i} x^{b_j} = x^{a_i+b_j} = x^{k}\). Therefore, \(c_k\) will be the number of such pairs \(a_i + b_j = k\).
Problem 8-4. Comparing Polynomials

Two single variable degree-\(d\) polynomials, \(P\) and \(Q\), with coefficients from \(\mathbb{Z}_p\) are said to be identical if \(P(x) = Q(x)\) for all \(x \in \mathbb{Z}_p\). Suppose you want to determine if two degree-\(d\) polynomials, \(F, G\) with coefficients in \(\mathbb{Z}_p\) are identical, where \(p > 2d\). However, you are not explicitly given \(F\) or \(G\). Rather, you are given two black boxes, which on any input \(x \in \mathbb{Z}_p\) return \(F(x)\) and \(G(x)\), respectively.

Give an efficient Monte-Carlo algorithm to determine if \(F\) and \(G\) are identical. If \(F = G\), your algorithm should output the correct answer with probability 1. If \(F \neq G\), your algorithm should output the correct answer with probability at least \(3/4\).

You may use the following fact: A degree-\(d\) non-zero polynomial has at most \(d\) values of \(x\) for which it evaluates to 0.

Solution: If \(F\) and \(G\) are the same polynomials, then \(F - G\) is the zero polynomial. Otherwise, it is 0 for at most \(d\) values of \(x\) in \(\mathbb{Z}_p\). Thus, if \(F \neq G\), and we choose a random point \(x\) in \(\mathbb{Z}_p\), then the probability that \(F(x) - G(x) = 0\) is at most \(d/p < 1/2\).

So our algorithm is as follows: Pick two random points \(x\) and \(y\) in \(\mathbb{Z}_p\) and evaluate \(F(x) - G(x)\) and \(F(y) - G(y)\). If both answers are 0, we output “\(F = G\). Otherwise, we output \(F \neq G\).

Note that if \(F = G\), then out algorithm always outputs the correct answer. If \(F \neq G\), then \(F(x) - G(x) = 0\) for a random \(x\) in \(\mathbb{Z}_p\) with probability at most \(1/2\), and \(F(y) - G(y) = 0\) with probability at most \(1/2\) so we output \(F = G\), i.e. the wrong answer, with probability at most \(1/4\).