CS60094 Computational Number Theory, Spring 2016–2017

Class Test

13–April–2017 | CSE 119/120, 6:30–7:30pm | Maximum marks: 30

Roll no: ________________ **Name:** ____________________________________

[*Write your answers in the question paper itself. Be brief and precise. Answer all questions.*]

1. Consider the extension field $\mathbb{F}_{3^n} = \mathbb{F}_3(\theta)$ with $f(\theta) = 0$, where $f(x) \in \mathbb{F}_3[x]$ is a monic irreducible polynomial of degree n.

(a) Let α be an element of $\mathbb{F}_{3^n}$ in this representation. Prove that $\alpha^{3^{n-1}}$ is the unique cube root of α in $\mathbb{F}_{3^n}$. How much time does it take to compute this cube root $\sqrt[3]{\alpha}$ using this exponentiation? **(4+4)**

Solution By Fermat's little theorem, $\alpha^{3^n} = \alpha$, that is, $\left(\alpha^{3^{n-1}}\right)^3 = \alpha$. In order to prove the uniqueness, let $\beta \in \mathbb{F}_{3^n}$ be a cube root of α. We have $\beta^3 = \alpha$, that is, $(\beta^3)^{3^{n-1}} = \alpha^{3^{n-1}}$, that is, $\beta^{3^n} = \alpha^{3^{n-1}}$, that is, $\beta = \alpha^{3^{n-1}}$.

Running time: The exponent 3^{n-1} contains about $(n-1)\log_2 3 = \Theta(n)$ bits. A repeated square-and-multiply algorithm requires $\Theta(n)$ iterations. Each iteration involves a squaring and a conditional multiplication of polynomials of degrees $\leqslant n-1$, each followed by reduction modulo the defining polynomial f. Using school-book arithmetic, this can be done in $O(n^2)$ time. To sum up, this exponentiation takes $O(n^3)$ time.

(b) We want to improve the running time to better than the exponentiation-based algorithm of Part (a). We precompute the two field elements $\theta^{1/3} = \theta^{3^{n-1}}$ and $\theta^{2/3} = (\theta^{1/3})^2$. Explain how we can express $\alpha = A_0(\theta^3) + A_1(\theta^3)\theta + A_2(\theta^3)\theta^2$ for polynomials A_0, A_1, A_2 over $\mathbb{F}_3$. Propose an efficient algorithm to compute $\sqrt[3]{\alpha}$ using this expression of α. What is the running time of your algorithm (excluding the time for precomputation)? **(4+4+4)**

Solution We have

$$\begin{aligned}\alpha &= a_0 + a_1\theta + a_2\theta^2 + \cdots + a_{n-1}\theta^{n-1} \\ &= (a_0 + a_1\theta^3 + a_2\theta^6 + \cdots) + (a_1 + a_4\theta^3 + a_7\theta^6 + \cdots)\theta + (a_2 + a_5\theta^3 + a_8\theta^6 + \cdots)\theta^2,\end{aligned}$$

so we take

$$\begin{aligned}A_0(\theta) &= a_0 + a_3\theta + a_6\theta^2 + \cdots, \\ A_1(\theta) &= a_1 + a_4\theta + a_7\theta^2 + \cdots, \\ A_2(\theta) &= a_2 + a_5\theta + a_8\theta^2 + \cdots.\end{aligned}$$

This gives

$$\sqrt[3]{\alpha} = \alpha^{3^{n-1}} = A_0(\theta^{3^n}) + A_1(\theta^{3^n})\theta^{3^{n-1}} + A_2(\theta^{3^n})(\theta^{3^{n-1}})^2 = A_0(\theta) + A_1(\theta)\theta^{1/3} + A_2(\theta)\theta^{2/3}.$$

Running time: The three polynomials A_0, A_1, A_2 can be prepared in $O(n)$ time. Next, we make two multiplications by the two precomputed elements $\theta^{1/3}$ and $\theta^{2/3}$; these take $O(n^2)$ time. Finally, the sums can be done in $O(n)$ time. So the overall running time is $O(n^2)$.

2. You are given a positive integer l. Your task is to find the largest l-bit prime. Propose an efficient sieving algorithm to solve this problem. Deduce the running time of your algorithm. **(5+5)**

Solution The largest l-bit number is $2^l - 1$. We maintain an array A of length $\lambda = \Theta(l)$ with the array index $i \in [0, \lambda - 1]$ standing for the l-bit number $2^l - 1 - i$. We initialize $A[i] = 1$ for all $i = 0, 1, 2, \ldots, \lambda - 1$. We also choose the first t primes $p_1, p_2, \ldots, p_t$. For each $j \in \{1, 2, 3, \ldots, t\}$, we first compute $r = (2^l - 1) \operatorname{rem} p_j$, and keep on setting $A[r] = 0$ and updating $r := r + p_j$ so long as $r < \lambda$.

When this sieving is done for all of the t small primes, we test the primality of $2^l - 1 - i$ for increasing values of i for which $A[i] = 1$. The first prime detected is returned.

If no prime is found in the range $[2^l - \lambda, 2^l - 1]$, we increase λ, and repeat. If λ is a small multiple of l, the chance of this is low. In case failure happens, we do not need to seed the interval $[2^l - \lambda, 2^l - 1]$ again (for the old λ), and can sieve the interval $[2^l - 2\lambda, 2^l - \lambda - 1]$.

Running time: Let us assume that the first sieve succeeds in identifying the largest l-bit prime. Initializing A takes $\Theta(\lambda) = \Theta(l)$ time. Each division of $2^l - 1$ by a small (single-precision) prime can be done in $\Theta(l)$ time. Marking cells of A takes a total of $\lambda \sum_{j=1}^{t} \frac{1}{p_j} = O(\lambda \ln \ln t)$ time. So the total running time of the sieving stage is dominated by the l remainder calculations, and is $\Theta(l^2)$.

Let $m = p_1 p_2 \ldots p_t$. Only those integers $2^l - 1 - i$ coprime to m have $A[i] = 1$ at the end of sieving, and their count is approximately $\lambda \phi(m)/m = \lambda (p_1 - 1)(p_2 - 1) \cdots (p_t - 1)/(p_1 p_2 \cdots p_t) \leqslant \lambda (1/2)(2/3) \cdots (t/(t+1)) = \lambda/(t+1)$. Each primality test takes time $O(l^3)$ using the Miller–Rabin algorithm. So the running time of this stage is $O(l^4/t)$.