Mid-Term Exam

(*reference answers*)

Prof. Manisha Kulkarni, Prof. G.Srinivasaraghavan

On: March 10, 2023	**Time**: 2 Hrs	**Max Marks**: 30

1 Theory

Q-1: Let p be an odd prime number. Prove using Fermat's Little Theorem that every prime divisor of $2^p - 1$ is greater than p.

Max Marks: 5

Answer: Let $q \leq p$ be an odd prime (trivially 2 cannot be a factor of $2^p - 1$). By FLT we have $2^{q-1} = 1 \bmod q$. Note that 2 is co-prime to q. Therefore the multiplicative order k of 2 in Z_q is such that $1 < k \leq (q-1)$. Moreover we know that when $a^i = 1 \bmod n$ for some a with multiplicative order l and co-prime to n then $l|i$. Now if $q|(2^p - 1)$ then $2^p \equiv 1 \bmod q$. This implies that $k|p$ which contradicts the fact that p is a prime. What happens to this argument when $q > p$? ■

Q-2: Let p be a prime such that $p > 10$. Find an integer $1 \leq N \leq 1000$ that satisfies the equation $5^{4p} \equiv N \mod (12p)$.

Max Marks: 5

Answer: Clearly $gcd(5, 12p) = 1$ for any $p > 10$ — only prime factors of $12p$ are $2, 3, p$. From Euler's Theorem we have that $5^{\phi(12p)} \equiv 1(\mod 12p)$. Also $\phi(12p) = 12p(1 - 1/p)(1 - 1/2)(1 - 1/3) = 4p - 4$. Therefore

$$5^{4p-4} \equiv 1(\mod 12p) \Rightarrow 5^{4p} \equiv 5^4(\mod 12p) \equiv 625(\mod 12p)$$

■

Q-3: It follows from the Chinese Remainder Theorem that there is an isomorphism

$$\phi : \frac{Z}{20Z} \rightarrow \frac{Z}{4Z} \times \frac{Z}{5Z}$$

In this case what is $\phi^{-1}(1, 3)$?

Max Marks: 5

Answer: ϕ is the Chinese Remainder map and $\phi^{-1}(1, 3)$ is the solution to the family of modular equations below:

$$a = 1 \mod 4$$
$$a = 3 \mod 5$$

Note that $4 * 5 = 20$. Let $n = 20, n_1 = 4, n_2 = 5, a_1 = 1, a_2 = 3$. Then $\left(\frac{n}{n_1}\right) = n_1^* = 5$ and therefore $n_1^* * 1 = 1 \bmod n_1$. So $1 = (n_1^*)^{-1} \bmod n_1$. Similarly $n_2^* = 4$ and $4 = (n_2^*)^{-1} \bmod n_2$. Therefore by the Chinese Remainder Theorem the solution to the modular equations is

$$\begin{aligned}\phi^{-1}(1,3) &= ((n_1^*)^{-1} \mod n_1) * n_1^* * a_1 + ((n_2^*)^{-1} \mod n_2) * n_2^* * a_2 \\ &= 1 * 5 * 1 + 4 * 4 * 3 \\ &= 53 = 13 \mod 40\end{aligned}$$

■

Q-4: Let p and q be distinct odd primes. Show that $p^{q-1} + q^{p-1} \equiv 1 \mod pq$.

Max Marks: 5

Answer: From Fermat's little theorem

$$\begin{aligned}p^{q-1} &\equiv 1 \mod q \Rightarrow p^q \equiv p \mod pq \\ q^{p-1} &\equiv 1 \mod p \Rightarrow q^p \equiv q \mod pq\end{aligned}$$

Adding the above two equations we get

$$\begin{aligned}p^q + q^p &\equiv (p+q) \mod pq \\ (p^{q-1} + q^{p-1})(p+q) - pq(p^{q-2} + q^{p-2}) &\equiv (p+q) \mod pq \\ (p^{q-1} + q^{p-1})(p+q) &\equiv (p+q) \mod pq \\ p^{q-1} + q^{p-1} &\equiv 1 \mod pq\end{aligned}$$

In the last step we have used the fact that $\texttt{gcd}(p+q, pq) = 1$ when p, q are primes. ■

2 Algorithms

Q-1: Consider the following algorithm ($\texttt{len}_2(n)$ denotes the size of the binary representation of n, though the actual representation for implementation may not necessarily be binary):

Initialize
$$k \leftarrow \left\lfloor \frac{\texttt{len}_2(n) - 1}{2} \right\rfloor, \quad m \leftarrow 2^k$$
for $i = (k-1)$ **downto** 0 **do**
  if $(m + 2^i)^2 \leq n$ **then**
    $m \leftarrow m + 2^i$
  end if
end for
return m

(a) Show (using appropriate assertions) that this algorithm correctly computes $\lfloor \sqrt{n} \rfloor$.

Max Marks: 4

(b) Show how this algorithm can be implemented in time $O(\texttt{len}(n)^2)$.

Max Marks: 3

Answer:

1. We show the correctness of the algorithm by employing appropriate invariant assertions. We will in fact prove something more general — that almost the same algorithm works for computing $\lfloor n^{1/e} \rfloor$ for any $e \geq 2$. Setting $e = 2$ in the version of the algorithm below will prove the correctness of the original algorithm.

Initialize
$$k \leftarrow \left\lfloor \frac{\texttt{len}_2(n) - 1}{e} \right\rfloor, \quad m \leftarrow 2^k$$
$\rightarrow$`Assertion 1`: $m = 2^k \leq n^{1/e} < 2^{k+1} \Rightarrow \lfloor n^{1/e} \rfloor$ is a $(k+1)$-bit number with no leading 0's
i.e., $n^{1/e} = 2^k + \delta$ for some $\delta < 2^k$
for $i = (k-1)$ **downto** 0 **do**
  if $(m + 2^i)^e \leq n$ **then**
    $m \leftarrow m + 2^i$
  end if
$\rightarrow$`Assertion 2`: $m \leq n^{1/e} < (m + 2^i)$
end for
return m

Proof of Assertion 1:

$$\begin{aligned}
\left\lfloor \frac{\mathtt{len}_2(n)-1}{e} \right\rfloor \leq \frac{\mathtt{len}_2(n)-1}{e} & \quad < \left\lfloor \frac{\mathtt{len}_2(n)-1}{e} \right\rfloor + 1 \quad \text{(from the definition of } \lfloor . \rfloor\text{)}\\
k \leq \frac{\mathtt{len}_2(n)-1}{e} & \quad < k+1\\
ek \leq (\mathtt{len}_2(n)-1) & \quad < e(k+1)\\
2^{ek} \leq 2^{\mathtt{len}_2(n)-1};\ \ \mathtt{len}_2(n) & \quad \leq e(k+1)\\
m^e = (2^k)^e \leq 2^{\mathtt{len}_2(n)-1};\ \ 2^{\mathtt{len}_2(n)} & \quad \leq (2^{k+1})^e\\
m^e = (2^k)^e \leq 2^{\mathtt{len}_2(n)-1} \leq n < 2^{\mathtt{len}_2(n)} & \quad \leq (2^{k+1})^e\\
m \leq n^{1/e} & \quad < 2^{k+1}
\end{aligned}$$

Note that when $i = 0$ (last iteration) Assertion 2 guarantees that $m \leq n^{1/e} < (m+1)$ which implies from the definition of $\lfloor . \rfloor$ that $m = \lfloor n^{1/e} \rfloor$.

Proof of Assertion 2: Let the binary representation of $\lfloor n^{1/e} \rfloor$ be $(1, b_{k-1}, b_{k-2}, ..., b_0)$. So $\lfloor n^{1/e} \rfloor = 2^k + \sum_{i=0}^{k-1} b_i 2^i$. The algorithm starts with $m = 2^k$ as in Assertion 1 and adds 2^i to m if $b_i == 1$, starting from b_{k-1} till b_0. It is convenient to subscript m with the iteration index i for the proof — let's denote the value of m at Assertion 2 in iteration i as m_i. So the assertion we need to prove is $m_i \leq n^{1/e} < (m_i + 2^i)$. The induction hypothesis implies $m_{i+1} \leq n^{1/e} < (m_{i+1} + 2^{i+1})$. Note that Assertion 1 is in fact the base case with $i = k$. There are two cases (step 3):

$\boldsymbol{(m_{i+1} + 2^i)^e \leq n}$: In this case $m_i = m_{i+1} + 2^i$ (step 4). Therefore trivially $m_i \leq n^{1/e}$ (the case condition). Also $m_i + 2^i = m_{i+1} + 2^i + 2^i = m_{i+1} + 2^{i+1} > n^{1/e}$ (induction).

$\boldsymbol{(m_{i+1} + 2^i)^e > n}$: Here $m_i = m_{i+1}$. Therefore $n^{1/e} < m_{i+1} + 2^i = m_i + 2^i$ (the case condition) and $m_i = m_{i+1} \leq n^{1/e}$ (induction).

2. The following is the implementation version of the algorithm — this is only for $e = 2$. Here the variables are subscripted with the iteration - for example m_i refers to the value of m at the end of iteration i. Steps 4,6 clearly take constant time. The only step with

---

1: Initialize

$$k \leftarrow \left\lfloor \frac{\mathtt{len}_2(n)-1}{2} \right\rfloor, \quad m_k \leftarrow 2^k$$

2: **for** $i = (k-1)$ **downto** 0 **do**

3:     **if** $\left(m_j + 2^{(j-1)}\right)^2 \leq n$ **then**   ▷ Taking $i+1$ as j; $(i+1)$ is the previous iteration

4:         $m_{(j-1)} \leftarrow m_j + 2^{(j-1)}$

5:     **else**

6:         $m_{(j-1)} = m_j$

7:     **end if**

8: **end for**

9: **return** m

---

non-trivial complexity in every iteration is step 3, to compute $(m_j + 2^{j-1})^2$. From previous iteration we would have $m_j = m_{(j+1)} + 2^j$ and we would have already computed $m_j^2 = \left(m_{(j+1)} + 2^j\right)^2$. The idea is to remember the value of m_j^2 from the earlier iteration and exploit the fact that $(m_j + 2^{(j-1)})^2 = m_j^2 + 2^j m + 2^{2(j-1)}$ where for m_j^2 on the RHS we simply recall the value of m_j^2 stored from the previous iteration. The second term on the RHS can be implemented in time $O(\texttt{len}(n))$ using bit-shifts (it is a multiplication by a power of 2). All additions are $O(\texttt{len}(n))$. The number of iterations that the loop in steps 2–8 will execute is also $O(\texttt{len}(n))$. The total running time is therefore $O((\texttt{len}(n))^2)$.

■

Q-2: Show that if $m = m_1.m_2 \dots m_k$, a product of k integers $m_i, k \geq 2$, then m can be computed using $O\left((\log m)^2\right)$ bit operations, independently of k.

Max Marks: 3

Answer: We prove this by induction on k. It is of course trivially true for $k = 1$. Suppose $k = 2$. Time to compute $m = m_1 m_2$ is $O((\log m_1)(\log m_2)) = O\left((\log m)^2\right)$. For any other $k > 2$, as the induction hypothesis, assume that the assertion holds for $1, \dots, (k-1)$. For k let $m = m_1.m_2 \dots m_k = M_{1l} M_{l+1,k}$ where M_{ij} for $i \leq j$ represents the product $m_i.m_{i+1} \dots m_j$. From the induction hypothesis, time to compute M_{1l} recursively is $O\left((\log M_{il})^2\right)$ and for $M_{l+1,k}$ it is $O\left((\log M_{l+1,k})^2\right)$. Time to compute m is therefore

$$\begin{aligned} O\left((\log M_{il})(\log M_{l+1,k}) + (\log M_{il})^2 + (\log M_{l+1,k})^2\right) &= O\left((\log M_{il} + \log M_{l+1,k})^2\right) \\ &= O\left((\log (M_{il}.M_{l+1,k}))^2\right) \\ &= O\left((\log m)^2\right) \end{aligned}$$

■