Solution Homework #1 - Prerequisites

Cryptography and Security 2020

Solution 1 Probabilities

Reliability of the test

Let T be the event *the test is positive* and D be the event *the patient has the disease.* We know the following

$$\Pr[T|D] = \frac{99}{100}$$
$$\Pr[T|\overline{D}] = \frac{1}{100}$$
$$\Pr[D] = \frac{1}{100}$$

and we are interested in

$$\Pr[D|T] = \frac{\Pr[T|D]\Pr[D]}{\Pr[T]} = \frac{\Pr[T|D]\Pr[D]}{\Pr[T|D]\Pr[D] + \Pr[T|\overline{D}]\Pr[\overline{D}]} = \frac{\frac{99}{100}\times\frac{1}{100}}{\frac{99}{100}\times\frac{1}{100} + \frac{1}{100}\times\frac{99}{100}} = \frac{1}{2} \; .$$

Thus, the probability of actually having the disease is $\frac{1}{2}$.

A new test

Let N be the number of old tests being positive. The probability to output a false positive is $\Pr[N \geq \frac{n}{2}|\overline{D}]$. Now, $\Pr[N = k|\overline{D}]$ is the probability that k out of n tests output false positive. Thus,

$$\Pr\left[N = k|\overline{D}\right] = \binom{n}{k} p^k (1-p)^{n-k} \; ,$$

where $p = \Pr[T|\overline{D}]$, since the output of the tests are independent conditioned on the even $\overline{D}$. Therefore, $N|\overline{D}$ follows a binomial distribution and $\mathbb{E}[N|\overline{D}] = np = \frac{n}{100}$. Following Markov's inequality, we obtain

$$\Pr\left[N \geq \frac{n}{2}|\overline{D}\right] \leq \frac{\mathbb{E}[N|\overline{D}]}{n/2} = \frac{2}{100} \; .$$

The bound is twice the probability to obtain a false positive with a single test, even though we added redundancy in this new test, thus it does not seem tight when n increases. One can also compute the value $\Pr\left[N \geq \frac{n}{2}|\overline{D}\right]$ for some value of n. For instance, for $n = 4$ and $n = 10$, we obtain respectively ≈ 0.0006 and $\approx 2.42 \times 10^{-8}$. In general, we observe that when n grows, the probability of having a false positive goes to zero, thus the bound is not tight.

Solution 2 Euclidean Domains

1. **Polynomial Rings**
For the first part, for all $f \in K[x]$, let $d(f)$ be the degree of f. It is fairly easy to check the properties $1, 2, 3$ for this function.
For the second part you can use Question 2.2. consider the ideal $\langle x_1, x_2 \rangle \leq K[x_1, x_2]$. This ideal can not be generated by a single element. One can also argue that there are no such m, r such that $x_1 = mx_2 + r$, with $d(r) < d(x_2)$. This would mean that $m = 0$ so either $r = 0$ which would mean $x_1 = 0 \times x_2 + 0 = 0$ which does not hold, or $d(r) = d(x_1 - 0 \times x_0) = d(x_1) < d(x_2)$. But using the same argument $d(x_2) < d(x_2)$ if we switch the roles of x_1 and x_2 which is a contradiction.

2. **PI Property**
Let $I \leq R$ be an ideal of R. Take $a \in I$ such that $d(a)$ is minimum in I. This element always exists, as the set $d(I)$ is discrete and has a lower bound (0). We have to prove that $\langle a \rangle = I$. It is obvious that $\langle a \rangle \subseteq I$ as $a \in I$. Now imagine $b \in I \backslash \langle a \rangle$. Due to the property 1, there are m and r such that $b = am + r$ and $d(r) < d(a)$. As I is an ideal and $a \in I$, am is also in I and $b - am$ is also in I as b and am are in I, which means $r \in I$ and $d(r) < d(a)$, which contradicts with how we selected a.

3. **GCD**
To prove the first, we have to prove $\langle a, b \rangle = \langle GCD(a, b) \rangle$, where $GCD(a, b)$ is the normal gcd in $\mathbb{Z}$. First we have that $a \in \langle GCD(a, b) \rangle$ and $b \in \langle GCD(a, b) \rangle$, as the gcd divides both a and b, which means $\langle a, b \rangle \subseteq \langle GCD(a, b) \rangle$. Also we have $\exists x, y \in \mathbb{Z}$ s.t. $ax + by = GCD(a, b)$. From this we have, $GCD(a, b) \in \langle a, b \rangle$, hence $\langle GCD(a, b) \rangle \subseteq \langle a, b \rangle$. So the definition is compatible.

   For part 2, we perform the usual Euclidean algorithm but this time using rule 1. We let $a_0 = a$ and $b_0 = b$, where a, b are the inputs of the algorithm. At each step for (a_n, b_n), we find m, r such that $a_n = b_n m + r$, and if $r = 0$ we output b_n, otherwise we take the tuple $(a_{n+1} = b_n, b_{n+1} = r)$ as our next output. Now $d(b_0) > d(b_1) > d(b_2) > d(b_3) > \dots$, so the sequence $d(b_i)$ is decreasing, but as it is always positive, at some point it should stop. So at a step ℓ, $a_\ell = b_\ell \times m + 0$. By backtracking the steps we get that $a \in \langle b_\ell \rangle$ and $b \in \langle b_\ell \rangle$, and also $b_\ell \in \langle a, b \rangle$, Which proves that it is in fact the GCD.

   To observe this, we have that $a_\ell = b_\ell m + 0$. This means that $a_\ell \in \langle b_\ell \rangle$. In the previous step $b_{\ell-1} = a_\ell$ and $a_{\ell-1} = b_{\ell-1} m' + b_\ell$ due to how the algorithm works. This means $a_{\ell-1} = a_\ell m' + b_\ell$. Now both a_ℓ and b_ℓ are in $\langle b_\ell \rangle$ so $a_{\ell-1}$ is also in $\langle b_\ell \rangle$. By continuing this we get $a_i, b_i \in \langle b_\ell \rangle$ for all $i \in \{0, \dots, \ell\}$, which means $a, b \in \langle b_\ell \rangle$.

Solution 3 Mastering recursivity

1. Recall that $x^{\log_z(y)} = y^{\log_z(x)}$ for all $x, y \in \mathbb{R}_{>0}$ whose logarithm in base z is well-defined. We assume that $\log(\cdot)$ denotes the binary logarithm. For all $k \in \mathbb{N}$ and $n = 2^k$, we have

$$T(2^k) \leq b^k T(1) + \sum_{0 \leq j < k} b^j S(2^{k-j}) \leq d2^{k \log b} + S(2^k) \sum_{0 \leq j < k} (b/c)^j = dn^{\log b} + S(n) \sum_{0 \leq j < k} (b/c)^j,$$

where the inequalities follow by induction and by $S(2^{k-j}) \leq c^{-j} S(2^k)$. Then,

$$\sum_{0 \leq j < k} (b/c)^j = \begin{cases} k = \log n & \text{if } b = c, \\ \frac{(b/c)^k - 1}{(b/c) - 1} = \frac{c}{b-c} \left(2^{k \log(b/c)} - 1\right) & \text{if } b \neq c. \end{cases}$$

2. Assume that $n = 2^k$. By assumption on S, we have $0 < S(1) \leq c^{-1}S(2) \leq \cdots \leq c^{-k}S(2^k)$. Now, the codomain of S is $\mathbb{N}$ and S is non-decreasing, whence

$$dn^{\log b} \leq dS(1)n^{\log b} \leq dc^{-k}S(2^k)n^{\log b} = dS(n)n^{\log b/c}.$$

For $n = 2^k$, we deduce by the previous point that

$$T(n) \leq \begin{cases} \left(\frac{d}{\log n} + 1\right) S(n)\log n & \text{if } b = c, \\ \left(d + \frac{c}{b-c}\right) S(n)n^{\log(b/c)} & \text{if } b > c. \end{cases}$$

For an arbitrary integer $n \in \mathbb{N}$, we let $k = \lceil \log n \rceil$. Assume that $b > c$ and let $\alpha = d + \frac{c}{b-c}$. Recall that S and T are non-decreasing and $S(2n) \leq \varepsilon S(n)$ for all $n \in \mathbb{Z}_+$. Thus, for sufficiently large n, we have

$$T(n) \leq T(2^k) \leq \alpha S(2^k)2^{k\log(b/c)} \leq \alpha S(2n)(2n)^{\log(b/c)} \leq 2\alpha\varepsilon S(n)n^{\log(b/c)}.$$

Assume that $b = c$ and let $\beta = d + 1$. For sufficiently large n, we have

$$T(n) \leq T(2^k) \leq (d/k + 1)\, S(2n)\log(2n) \leq 2\beta\varepsilon S(n)\log n.$$

3. Observe that $fg = F_1G_1x^n + (F_1G_0 + F_0G_1)x^{n/2} + F_0G_0 = h$. The correctness then follows by induction on k for $n = 2^k$ and by the fact that the algorithm terminates.

4. Since f and g have degrees at most n, polynomials F_i and G_i have degrees at most $n/2$. Therefore, Karatsuba's algorithm requires three calls to itself on polynomials of degree at most $n/2$ and

   (a) $n = n/2 + n/2$ additions for computing $F_0 + F_1$ and $G_0 + G_1$,

   (b) $2n = n + n$ subtractions to compute $h_2 - h_1 - h_0$,

   (c) n additions for adding $((F_0 + F_1)(G_0 + G_1) - F_0G_0 - F_1G_1)x^{n/2}$ to $F_1G_1x^n + F_0G_0$.

   Denote by $T(n)$ the complexity of Karatsuba's algorithm. By defining $(b, c, d) = (3, 2, 1)$ and $S(n) = 4n$, we deduce by the previous points that

$$T(n) \leq n^{\log 3} + 2S(n)\left(n^{\log 3 - 1} - 1\right) = 9n^{\log 3} - 8n.$$

5. Since $\log 3 < 1.59$, we have $9n^{\log 3} - 8n < 9n^{1.59} = O(n^{1.59})$. This can also be shown using the asymptotic relation since $T(n) = O(4n \cdot n^{\log 3/2}) = O(n^{\log 3}) = O(n^{1.59})$.