MTAT.07.003 Cryptology II
Spring 2012 / Exercise session ?? / Example Solution

Exercise (From expected to strict running time). *Let $\mathbb{G}$ be a finite q-element group such that all elements $y \in \mathbb{G}$ can be expressed as powers of $g \in \mathbb{G}$. Let $\mathcal{A}$ be an algorithm that always finds a discrete logarithm with the expected running-time at most τ. Construct a t-time algorithm $\mathcal{B}$ that fails with probability 2^{-n} and its running-time t is linear in τ and in n.*

Solution. Let t be a function mapping randomness of the algorithm $\omega \in \Omega$ and its input $y \in \mathbb{G}$ to the running time of the algorithm, i.e., $t(y;\omega)$ is the running time of $\mathcal{A}$ on input y and randomness ω. From the assumption, we know that

$$\forall y \in \mathbb{G} : \quad \mathop{\mathbf{E}}_{\omega \in \Omega}[t(y;\omega)] \leq \tau \enspace .$$

By using Markov's inequality, we get that the probability that algorithm $\mathcal{A}$ runs longer than t_0 steps is

$$\Pr\left[t(y;\omega) \geq t_0\right] \leq \frac{\mathbf{E}_{\omega\in\Omega}[t(y;\omega)]}{t_0} \leq \frac{\tau}{t_0} \enspace .$$

Let us consider the probability that the algorithm fails provide an output in time 2τ. The inequality derived above allows us to punt the corresponding probability form above:

$$\forall y \in \mathbb{G} : \quad \Pr\left[t(y;\omega) \geq 2\tau\right] \leq \frac{\tau}{2\tau} = \frac{1}{2} \enspace .$$

Now let $\mathcal{A}_{2\tau}$ be an algorithm that invokes $\mathcal{A}$ and waits its output for exactly 2τ time. If $\mathcal{A}$ succeeds, it outputs $\mathcal{A}$'s output and $\perp$ otherwise. It is easy to construct such an algorithm from the code of $\mathcal{A}$ by replacing each instruction of $\mathcal{A}$ by a set of instructions: we first check if a dedicated time variable is smaller than 2τ, then we execute the instruction of $\mathcal{A}$, finally we increment the dedicated time variable t by 1. If $\mathcal{A}$ and $\mathcal{A}_{2\tau}$ are random access machines, then it is easy to see that the running time of $\mathcal{A}_{2\tau}$ is $\mathrm{O}(2\tau)$. If $\mathcal{A}_{2\tau}$ is a Turing machine with an extra working tape compared to $\mathcal{A}$ then the same claim holds. However, if $\mathcal{A}$ and $\mathcal{A}_{2\tau}$ must be turing machines such that the number of working tapes is the same, we can only prove that $\mathcal{A}_{2\tau}$ runs in time $\mathrm{O}(\tau^2)$ because the location of the dedicated timer t might be $\Omega(\tau)$ apart from the symbol $\mathcal{A}_{2\tau}$ is modifying inside the instruction block. Regardless of the bound on the running time we have proven

$$\forall y \in \mathbb{G} : \quad \Pr\left[x \leftarrow \mathcal{A}_{2\tau}(y) : g^x \neq y\right] = \Pr\left[t(y;\omega) \geq 2\tau\right] \leq \frac{1}{2} \enspace .$$

Now, let us consider the construction

$$
\begin{array}{l}
\mathcal{B}^{\mathcal{A}_{2\tau}}(n,y) \\
\left[\begin{array}{l}
\text{For } \ell \in \{1,\ldots,n\} \text{ do} \\
\quad \left[\begin{array}{l} x \leftarrow \mathcal{A}_{2\tau}(y) \\ \text{if } g^x = y \ \ \mathbf{return}\ y \end{array}\right. \\
\mathbf{return}\ \perp
\end{array}\right.
\end{array}
$$

Clearly, algorithm $\mathcal{B}$ runs in time less than $c \cdot 2n\tau$ for some small overhead constant $c > 1$ in the Random Access Machine model. During this time, $\mathcal{B}$ makes at most n queries to $\mathcal{A}_{2\tau}$. As the probability of failure each time is $\frac{1}{2}$, then after n invocations the failure probability is $\left(\frac{1}{2}\right)^n = 2^{-n}$. Thus, we have constructed an algorithm $\mathcal{B}$ which runs in time $\mathrm{O}(n\tau)$ and its failure probability is 2^{-n}, as required.

For the Turing machines, the construction runs in time $\mathrm{O}(n\tau^2)$ and in general it is difficult if not impossible to show that the running rime can be actually bounded by $\mathrm{O}(n\tau)$. The only way to achieve that is to modify the definition of a Turing machine so that all algorithms can use timers. However, the latter is technically an non-trivial task, as the algorithm $\mathcal{A}$ might then already use a timer and we must makes sure that $\mathcal{A}_{2\tau}$ can make calls to timer without obligating timers used by $\mathcal{A}$.