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

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

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

By using Markov’s inequality, we get that the probability that algorithm $A$ runs longer than $t_0$ steps is

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

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

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

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

$$\forall y \in G : \Pr [x \leftarrow A_{2\tau}(y) : g^x \neq y] = \Pr [t(y; \omega) \geq 2\tau] \leq \frac{1}{2} .$$

Now, let us consider the construction

$$B^{A_{2\tau}}(n, y)$$

```plaintext
   For \( \ell \in \{1, \ldots, n\} \) do
     \( x \leftarrow A_{2\tau}(y) \)
     if \( g^x = y \) return \( y \)
   return \( \perp \)
```

Clearly, algorithm $B$ 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, $B$ makes at most $n$ queries to $A_{2\tau}$. As the probability of failure each time is $\frac{1}{2}$, then after $n$ invocations the failure probability is $(\frac{1}{2})^n = 2^{-n}$. Thus, we have constructed an algorithm $B$ which runs in time $O(n\tau)$ and its failure probability is $2^{-n}$, as required.

For the Turing machines, the construction runs in time $O(n\tau^2)$ and in general it is difficult if not impossible to show that the running time can be actually bounded by $O(n\tau)$. 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 $A$ might then already use a timer and we must makes sure that $A_{2\tau}$ can make calls to timer without obligating timers used by $A$. 

1