**CS1200: Intro. to Algorithms and their Limitations** Prof. Salil Vadhan

# Sender–Receiver Exercise 7: Reading for Senders

*Harvard SEAS - Fall 2025* *2025-11-20*

## 1 Is Linear Time is unsolvable

The goals of this exercise are:

- to develop your skills at understanding, distilling, and communicating proofs and the conceptual ideas in them,
- to practice reductions for proving unsolvability, and gain more intuition for what kinds of problems about programs are unsolvable.

Sections 1.1 is also in the reading for Receivers. Your goal will be to communicate the *proof* of Theorem 1.1 (i.e. the content of Section 1.2) to your Receiver. Section 1.3 contains questions for you and your Receiver to think about if you finish the exercise early; there is no need to prepare anything in advance for that.

### 1.1 The Result

Rice's Theorem (Theorem 25.16 in the textbook) says that all nontrivial problems about the input–output behavior of programs (i.e. about a program's semantics) are unsolvable. Here we will see an example of a computational problem that is *not* about the input–output behavior of programs but is nevertheless unsolvable:

**Input:** A RAM program $P$

**Output:** `yes` if $P$ has running time $O(n)$, `no` otherwise

**Computational Problem** Is Linear Time

As usual, the statement "$P$ has running time $O(n)$" means that there are constants $c$ and $n_0$ such that for all $n \geq n_0$ and all inputs $x$ of length at most $n$, $P(x)$ halts within $c \cdot n$ steps. Note that the constants $c$ and $n_0$ are allowed to depend on $P$.

**Theorem 1.1.** Is Linear Time *is unsolvable.*

### 1.2 The Proof

By Lemma 25.9 (using reductions to prove unsolvability) and Theorem 25.10 (Halts On Empty–RAM is unsolvable) in the textbook, it suffices to prove that Halts On Empty–RAM $\leq$ Is Linear Time. That is, we need to give an algorithm `Red` that can decide whether a program $P$ halts on $\varepsilon$ using an oracle for IS LINEAR TIME. As in the other examples we have seen, we will give a mapping reduction, following the template given in Algorithm 1.1

```
Red($P$):
Input   : A RAM program $P$
Output  : yes if $P$ halts on $\varepsilon$, no otherwise
0 Construct from $P$ a program $Q_P$ such that whether or not $Q_P$ runs in time $O(n)$ will tell
   us whether or not $P$ halts on $\varepsilon$;
1 Feed $Q_P$ to the IS LINEAR TIME oracle, and use the result to decide whether to output
   yes or no;
```

**Algorithm 1.1:** Template for HALTS ON EMPTY–RAM $\leq$ IS LINEAR TIME

How can we construct $Q_P$ from $P$? One idea is to have, for every input $x$ of length $n$, $Q_P(x)$ run $P$ on $\varepsilon$ for up to $n^2$ steps. That way, if $P$ does not halt on $\varepsilon$, then $Q_P$ takes time $\Omega(n^2)$, but if $P$ does halt on $\varepsilon$, then $Q_P$'s execution will stop earlier.

In more detail (but still pseudocode rather than formal RAM code):

```
$Q_P(x)$:
0 Let $n$ = input_len;
1 foreach $i = 0$ to $n - 1$ do
2     $M[i] = 0$
3 input_len = 0;
4 Run $P$ for upto $n^2$ steps (unless it halts sooner);
```

**Algorithm 1.2:** The RAM program $Q_P$ constructed from $P$

The commands of $Q_P$ before Line 4 are to set up the configuration of memory and `input_len` to correspond to input $\varepsilon$, so that we faithfully simulate $P$ on $\varepsilon$

**Claim 1.2.** *$Q_P$ runs in time $O(n)$ if and only if $P$ halts on $\varepsilon$.*

*Proof.* If $P$ does not halt on $\varepsilon$, then the execution of $P$ in Line 4 will always take at least $n^2$ steps, so $Q_P$ does not have runtime $O(n)$.

Conversely, suppose that $P$ does halt on $\varepsilon$, say in $T_P(\varepsilon)$ steps. Note that $T_P(\varepsilon)$ depends only on $P$, not on the input $x$ given to $Q_P$. Let's now analyze the runtime of $Q_P$ on an input $x$ of length $n$. The loop for erasing the input before running $P$ takes time $O(n)$. The simulation of $P$ in Line 4 will take time $O(n^2)$ if $n^2 < T_P(\varepsilon)$ and will take time $O(T_P(\varepsilon))$ if $n^2 \geq t_0(P)$, where the $O(\cdot)$ is to allow extra time to maintain a counter to make sure that we don't run $P$ for more than $n^2$ steps. Thus, if we set $n_0 = \sqrt{T_P(\varepsilon)}$, then for all $n \geq n_0$, the overall runtime of $Q_P(x)$ is

$$O(n) + O(T_P(\varepsilon)) = O(n) + O(1) = O(n),$$

which means that $Q_P$ runs in time $O(n)$. □

With this claim, we can fill in the details of our reduction from HALTS ON EMPTY to IS LINEAR TIME, given as Algorithm 1.3.

```
Red($P$):
Input   : A RAM program $P$
Output  : yes if $P$ halts on $\varepsilon$, no otherwise
0 Construct from $P$ the program $Q_P$ described in Algorithm 1.2;
1 Feed $Q_P$ to the IS LINEAR TIME oracle, and return whatever the oracle returns;
```

**Algorithm 1.3:** HALTS ON EMPTY–RAM $\leq$ IS LINEAR TIME

The correctness of the reduction follows from Claim 1.2, and thus we conclude that Is Linear Time is unsolvable.

**Tips for understanding the proof:**

1. Remember that we are looking at asymptotic runtime of $Q_P$ so think about what would happen when we fix $P$ and then take $n$ arbitrarily large.
2. You may find it helpful to sketch a graph of $Q_P$'s runtime when $P$ halts on $\varepsilon$ (say with $T_P(\varepsilon) = 1000$) and when it does not. What is the asymptotic behavior of each graph?
3. The Is Linear Time oracle only takes in a program as an input. We don't care how the oracle works (it is just assumed to solve the problem in one time step) but it might be helpful to imagine the oracle as running $Q_P$ on all (infinitely many) inputs $x$, graphing its worst-case runtime $T(n)$, and seeing if the function $T(n)$ is $O(n)$.

## 1.3 Food for Thought

If you and your partner(s) finish early, here are some additional questions or issues you can think about:

1. We constructed $Q_P$ so that $Q_P$ has running time $O(n)$ if and only if $P$ halts on $\varepsilon$. Can you think of how to construct $Q_P$ so that $Q_P$ has running time $O(n)$ if and only if $P$ *doesn't* halt on $\varepsilon$? If you used such a construction, how would the reduction in Algorithm 1.3 change?
2. So far, our intuition for unsolvability has been that it comes from the possibility that RAM programs don't halt. However, the programs $Q_P$ constructed in the above reduction always halt in time $O(n^2)$. Thus, the same reduction proves unsolvability of the following variant of Is Linear Time, where we *promise* that the input program halts in time $O(n^2)$:

   **Input:** A RAM program $P$ with running time $O(n^2)$

   **Output:** `yes` if $P$ has running time $O(n)$, `no` otherwise

   **Computational Problem** Is Linear Time With Promise

   Try to develop some of your own intuition for what makes a problem like this, on always-halting programs, unsolvable.