Individual: 7.2.25 (d, g, j)

7.2.5 (d) False: ACCEPTANCE is undecidable but is recognizable.

(g) True: if p recognizes $L_1$, and q recognizes $L_2$, on input i we can run p and q "in parallel," i.e.

\[
\begin{align*}
\text{Phase 1} & \quad - \text{simulate } p \text{ on input } i \text{ for } 1 \text{ step} \\
\text{Phase 2} & \quad - \quad q \quad \ldots \\
\text{Phase 3} & \quad - \text{simulate } p \quad \ldots \quad 2 \text{ steps} \\
\text{Phase 4} & \quad - \quad q \quad \ldots \quad 2 \text{ steps} \\
& \quad \quad \text{etc.,} \\
\end{align*}
\]

[If any phase ends early, then we proceed to the next phase.]

Whenever p or q accepts i, then we accept i.

If p or q accepts i, then the above algorithm eventually accepts i; if neither p nor q accepts i, then this algorithm never ends, so this algorithm does not accept i. Hence this algorithm
recognizes $L_1 \cup L_2$.

(i) False: ACCEPTANCE is recognizable, and 

$$\Sigma^* \setminus \text{ACCEPTANCE}$$

$$= \text{NON-ACCEPTANCE} \cup L$$

where

$$L = \left\{ \text{strings not of the form } p \sigma_0 i \right\}$$

$$= \left\{ \text{strings without a } \sigma_0 \right\}$$

$$\cup \left\{ p \sigma_0 i \mid p \text{ has no } \sigma_0 \text{ but is not a valid Python program} \right\}$$

which is decidable (see class on Sept 22)

So ACCEPTANCE $\cup L$ is recognizable (see part (g) above), but

NON-ACCEPTANCE = $\Sigma^* \setminus (\text{ACCEPTANCE} \cup L)$

is unrecognizable.
7.2.26 \((a, b, d, f)\) (Gray)

(a) **Decidable** (and therefore recognizable): a "debugging" or "universal" Python program can simulate \(p\) on input \(i\), for any finite number of steps.

(b) **Recognizable but undecidable**: recognizable since we can simulate \(p\) on input \(i\), and if the simulation stops then we accept \(p\sigma_i\) iff the simulation ends with \(p\) rejecting \(i\) (i.e. we negate the result of \(p\) on input \(i\)).

(We also reject any input not of the form \(p\sigma_i\).)

This algorithm accepts a string iff it is of the form \(p\sigma_i\) where \(p\) (is a valid Python program that rejects \(i\)).

**Undecidable**: if it were decidable, we could decide the language
\[ \text{ACCEPTANCE } = \{ \sigma \sigma_0 i \mid p \text{ accepts } i \} \]

by taking an input \( p \sigma_0 i \) with \( p \) a valid Python program, forming \( q \) that works like \( p \) but negates the result of \( p \), and then seeing if \( q \sigma_0 i \) lies in the language \( \{ q \sigma_0 i \mid q \text{ rejects } i \} \). This would decide \( \text{ACCEPTANCE} \), which is impossible.

(d) \underline{Unrecognizable}: Let \( L_1 \) denote this language, and \( L_2 = \{ \text{strings not of the form } p \sigma_0 i \text{ with } p \text{ a valid Python program} \} \).

If \( L_1 \) were recognizable, then since \( L_2 \) is recognizable, then \( L_1 \cup L_2 \) would be recognizable. But since
\[ \Sigma_{\text{ASCII}}^* \setminus (L_1 \cup L_2) = \text{ACCEPTANCE} \]

is recognizable, both

\[ L_1 \cup L_2 \text{ and } \text{ACCEPTANCE} \]

would be decidable \([\text{since in class we proved that } L \text{ and } \Sigma_{\text{ASCII}}^* \setminus L \text{ recognizable } \Rightarrow \text{they are both decidable}]. \]

This would imply that \( \text{ACCEPTANCE} \) is undecidable, which is impossible.

(f) Recognizable, since we can list the elements of \( \Sigma_{\text{ASCII}}^* \) as \( i_1, i_2, i_3, \ldots \).

We can run the algorithm

Phase 1: simulate \( p \) on \( i_1 \) for one step

Phase 2: \( \ldots \ldots \ldots \ldots \) 2 steps, and \( i_2 \) for one step

(See class notes and the next homework problems)
and stop at Phase $k$ and accept $p_{0,i}$ if two inputs $i_1, \ldots, i_k$ are accepted (in the number of steps allowed). This algorithm accepts $p_{0,i}$ iff $p$ accepts at least 2 of $i_1, i_2, \ldots$, i.e. at least 2 inputs.

**Undecidable**: If this language, $L$, were decidable, then the following algorithm would decide ACCEPTANCE (with yields a contradiction):

given an input:

1. check if it is of the form $p_{0,i}$ with $p$ a valid Python program, if so:

2. create a Python program $q$ that works like $p$ except that its input variable — once set by $p$ — is immediately
This program $q$ (which depends on $p$ and $i$)
- accepts all inputs if $p$ accepts $i$
- does not accept any input if $p$ does not accept $i$.

Hence

$q \in L \iff p$ accepts $i$.

Hence if $L$ is decidable, then there is a decider that recognizes ACCEPTANCE, which is impossible.
(2, Group Homework)

(a) If \( L \subseteq \Sigma_{\text{ASCII}}^* \) is recognizable, then in class we showed

\[ \Sigma_{\text{ASCII}}^* \setminus L \text{ is recognizable } \Rightarrow L \text{ is decidable} \]

Hence, the equivalent contrapositive statement is

\[ L \text{ is undecidable } \Rightarrow \Sigma_{\text{ASCII}}^* \setminus L \text{ is unrecognizable}. \]

(b) 7.2.26(g)

If \( L = \{ p \mid p \text{ accepts all inputs} \} \), then

\[ \Sigma_{\text{ASCII}}^* \setminus L = L_1 \cup L_2 \text{, where} \]

\[ L_1 = \{ p \mid p \text{ is a valid Python program} \mid p \text{ accepts at least one input} \} \]

= \text{ACCEPTS\_SOME\_INPUT} \text{ from class}

\[ L_2 = \{ p \mid p \text{ is not a valid Python program} \}, \text{which we explained is decidable in class} \]
If $L$ were recognizable, then since $L_1$ and $L_2$ are recognizable,

$$L \text{ and } \sum_{\text{All } w}^* \setminus L = L_1 \cup L_2 \text{ would both be recognizable,}$$

hence both would be decidable. Hence $L_1 \cup L_2$ would be decidable; since $L_2$ would be decidable, so would be $L_1$.

But the same argument as in 7.2.26(f) above shows that $L_1$ is undecidable.

Hence we would get a contradiction.

Hence $L$ is unrecognizable.
The input \( i_e \) would be accepted after \( m \) steps in Phase \( l+m \), but not before. For all \( k \in \mathbb{N} \), Phase \( k \) takes \( 1+2+\ldots+k \) steps \( = \binom{k+1}{2} \) steps.

Hence we run Phases \( 1, 2, \ldots, l+m-1 \) without an acceptance, and sometime during Phase \( l+m \) we stop. Hence the algorithm takes:

- at least \( \binom{2}{2} + \cdots + \binom{l+m}{2} \) steps

AND

- at most \( \binom{2}{2} + \cdots + \binom{l+m}{2} + \binom{l+m+1}{2} \) steps.

Since

\[
\sum_{k=2}^{l+m} \binom{k+1}{2} = \binom{l+m+1}{3} = \frac{(l+m+1)(l+m)(l+m-1)}{6}
\]

\[
= \frac{(l+m)^3 - (l+m)}{6} = \frac{1}{6} (l+m)^3 + O(1)(l+m),
\]

and since \( \binom{l+m+1}{2} = \frac{(l+m)^2 + (l+m)}{2} = O(1)(l+m)^2 \),
the total number of steps is bounded above by \( \frac{1}{6} (l+m)^2 + O(1) (l+m)^2 \)
and below by \( \frac{1}{6} (l+m)^2 + O(1) (l+m) \)

\[ = \frac{1}{6} (l+m)^2 + O(1) (l+m)^2. \]

(b) Similarly, this algorithm runs at least to all phases \( 1, \ldots, k-1 \), where \( k = \max(l,m) \) and some part of Phase \( k \) (or none of Phase \( k \))

Hence the number of steps is

\[ 1 + 2^2 + \ldots + (k-1)^2 + O(1) k^2 \]

\[ = \frac{(k-1)k(2k-1)}{6} + o(1) k^2 = \frac{1}{3} k^3 + O(1) k^2 \]

\[ = \frac{1}{3} \left( \max(l,m) \right)^3 + O(1) \left( \max(l,m) \right)^2 \]
(c) Similarly this algorithm runs for
\[
1 + 5^2 + (5-2)^2 + \ldots + (5(k-1))^2 + \mathcal{O}(k^2)
\]
\[
= 25 \left( \frac{1}{3} k^3 + \mathcal{O}(k^2) \right) + \mathcal{O}(k^2)
\]
\[
= \frac{25}{3} k^3 + \mathcal{O}(k^2)
\]
where \( k = \frac{\max(l,m)}{5} + \mathcal{O}(1) \).

Hence this algorithm runs for
\[
\frac{25}{3} \left( \frac{\max(l,m))^3}{125} + \mathcal{O}(1) \frac{\left( \max(l,m) \right)^2}{\max(l,m)}
\]
\[
= \frac{1}{3} \cdot \frac{1}{5} \left( \max(l,m) \right)^3 + \mathcal{O}(1) \left( \max(l,m) \right)^2
\]

(d) Similarly, for any \( k \in \mathbb{N} \), if Phase \( k \) runs
\( 1, \ldots, i_{B_k} \) for \( B_k \) steps,
the total number of steps is
\[ \frac{1}{3} \frac{1}{B} \left( \max(l, m) \right)^2 + O(1) \left( \max(l, m) \right)^2 \]

[You might notice that O(1) above depends on B, and, in fact, grows with B as B \to \infty...]

Hence, for any \( C > 0 \) we may choose \( B \) with \( \frac{1}{3B} < C \) to get an algorithm running in
\[ \leq C \left( \max(l, m) \right)^2 + O(1) \left( \max(l, m) \right)^2 \] steps.
(4c) Let Phase\(k\) run \(p\) on each of 
\(\{1, 2, \ldots, 2^k\}\) for \(2^k\) steps. Then this
algorithm runs at most \(\log_2(\max(l, m)) + 1\)
phases. Since Phase \(k\) takes \((2^k)^2\) steps, the total steps in Phases \(1, \ldots, k\) is

\[
1 + 2^2 + 4^2 + \ldots + (2^k)^2
\]

\[=
1 + 4 + 4^2 + \ldots + 4^k = \frac{4^{k+1} - 1}{4 - 1} = O(4^k).
\]

For \(k \leq \log_2(\max(l, m)) + 1\) we have 

\(2^k \leq 2 \max(l, m)\) and so \(4^k = O(1)(\max(l, m))^2\)

Hence the total number of steps needed
is \(O(1)(\max(l, m))^2\)
(4b) Let $B \in \mathbb{N}$, and consider inputs $i_1, i_2, \ldots, i_B$. The algorithm must at some point run $p$ on each of $i_1, i_2, \ldots, i_B$ for at least $B$ steps; let $i_a$, with $1 \leq a \leq B$, be the last element of $\{i_1, \ldots, i_B\}$ which the algorithm runs for at least $B$ steps. Then if $p$ accepts $i_a$ after $B$ steps, then the algorithm must have run for at least $B^2$ steps. Since

$$B = \max(a, B),$$

this algorithm has run for at least

$$\left(\max(a, B)\right)^2$$

steps. Hence for any value $B$, there is a
is a value of \( l,m \) such that

\[ B = \max(l,m) \quad (\text{namely } l=a, m=B \text{ above}), \]

such that the algorithm runs for at least \((\max(l,m))^2\) steps.

So you can take \( c=1 \) by this argument.

Can you find a value \( c>1 \) such the algorithm must run for at least \( c(\max(l,m))^2 \) steps, at least for infinity many values of \( B=\max(l,m) \)??

The algorithm in part (a) shows that

\[ c = 4 \times \frac{4}{3} = \frac{16}{3} \]

is the best possible lower bound...