Problem 1

We build a 3-tape Turing Machine M which decides the language

$$L \stackrel{def}{=} \{x\#y\#z : x, y, z \in \{0,1\}^* \text{ and } x + y = z \text{ when interpreted as binary numbers}\}$$

Informally, M works in 3 stages:

Stage 1 M copies x on its second tape, without any leading zeros, preceded by a left-end tape marker \$.

Stage 2 M copies y on its third tape, without any leading zeros, preceded by a left-end tape marker \$.

Stage 3 M moves to the right end of z on tape 1, of x on tape 2, and of y on tape 3. After that, it starts verifying that $x + y = z$ by basic addition with carry bit. We take special care to accomodate for the case where there are leading zeros in z.

Now on to the formal specification. To start the machine we have a transition which writes the \$ signs on tapes 2 and 3:

$$\delta(q_0, c, \sqcup, \sqcup) = (q_1, c, \$, \$, -, R, R), \forall c \in \{0, 1, \#\}$$

Right now head 1 is pointing at the beginning of x (it didn't move), while heads 2 and 3 are pointing at the blanks after the left-end tape markers \$. We begin *stage 1* described above.

$$\begin{aligned}
\delta(q_1, 0, \sqcup, \sqcup) &= (q_1, 0, \sqcup, \sqcup, R, -, -) \\
\delta(q_1, 1, \sqcup, \sqcup) &= (q_2, 1, 1, \sqcup, R, R, -) \\
\delta(q_1, \#, \sqcup, \sqcup) &= (q_3, \#, \sqcup, \sqcup, R, L, -) \\
\delta(q_2, c, \sqcup, \sqcup) &= (q_2, c, c, \sqcup, R, R, -), \forall c \in \{0, 1\} \\
\delta(q_2, \#, \sqcup, \sqcup) &= (q_3, \#, \sqcup, \sqcup, R, L, -)
\end{aligned}$$

While in q_1 we have seen only leading zeros from x and haven't copied anything. While in q_2, we have copied part of x on the second tape. When we enter q_3, head 1 is pointing at the beginning of y, head 2 is pointing at the last character of x, and head 3 is pointing at the blank after the \$. Remark that we allow for the possibility that the x we copy on tape 2 is the empty string. In that case, when entering q_3, head 3 is pointing at \$.

Next we move on to *stage 2*:

$$\begin{aligned}
\delta(q_3, 0, c, \sqcup) &= (q_3, 0, c, \sqcup, R, -, -), \forall c \in \{0, 1, \$\} \\
\delta(q_3, 1, c, \sqcup) &= (q_4, 1, c, 1, R, -, R), \forall c \in \{0, 1, \$\} \\
\delta(q_3, \#, c, \sqcup) &= (q_5, \#, c, \sqcup, R, -, L), \forall c \in \{0, 1, \$\} \\
\delta(q_4, c, c', \sqcup) &= (q_4, c, c', c, R, -, R), \forall c, c' \in \{0, 1, \$\} \\
\delta(q_4, \#, c, \sqcup) &= (q_5, \#, c, \sqcup, R, -, L), \forall c \in \{0, 1, \$\}
\end{aligned}$$

This stage is very similar to the former, so we will skip its detailed explanation.

Next we want to move to the right end of z on tape 1:

$$\begin{aligned}
\delta(q_5, c, c', c'') &= (q_5, c, c', c'', R, -, -), \forall c \in \{0, 1\}, \forall c', c'' \in \{0, 1, \$\} \\
\delta(q_5, \sqcup, c', c'') &= (q_6, \sqcup, c', c'', L, -, -), \forall c', c'' \in \{0, 1, \$\}
\end{aligned}$$

Finally, this is where the addition begins. In state q_6 we have no carry bit, while in state q_7 we do have a carry bit.

$$\begin{aligned}
\delta(q_6,0,c,c') &= (q_6,0,c,c',L,L,L), \forall c,c' \in \{0,\$\} \\
\delta(q_6,1,1,c) &= (q_6,1,1,c,L,L,L), \forall c \in \{0,\$\} \\
\delta(q_6,1,c,1) &= (q_6,1,c,1,L,L,L), \forall c \in \{0,\$\} \\
\delta(q_6,0,1,1) &= (q_7,0,1,1,L,L,L) \\
\delta(q_6,\#,\$,\$) &= (q_{accept},\ldots) \\
\delta(q_7,1,c,c') &= (q_6,1,c,c',L,L,L), \forall c,c' \in \{0,\$\} \\
\delta(q_7,0,1,c) &= (q_7,0,1,c,L,L,L), \forall c \in \{0,\$\} \\
\delta(q_7,0,c,1) &= (q_7,0,c,1,L,L,L), \forall c \in \{0,\$\} \\
\delta(q_7,1,1,1) &= (q_7,1,1,1,L,L,L)
\end{aligned}$$

This is it! Our machine has the states which appear above, $Q \stackrel{def}{=} \{q_0 \ldots q_7, q_{accept}, q_{reject}\}$, the tape alphabet is $\Gamma = \{0,1,\#,\$,\sqcup\}$, and the transition function is as described above, plus transitions to q_{reject} for anything not previously specified.

Problem 2

We assume the following behaviour for M: if $\delta(q,c) = (q',c',L)$ for some states q and q', and some tape symbols c and c', then

$$\forall w \in \Gamma^*, (q,\epsilon,cw) \to^M (q',\epsilon,c'w)$$

In other words, M changes state and writes the new character before attempting and failing to move the head left of the end of the tape.

Let us be given an arbitrary Turing Machine $M = (Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$. Our goal is to construct $M' = (Q',\Sigma',\Gamma',\delta',q'_0,q'_{accept},q'_{reject})$ such that $\forall w \in \Sigma^*, M'(w) = M(w) \in \{yes, no\}$, and such that M' never moves its head left of the end of the tape. M' will operate as follows:

We obviously use the same input alphabet:

$$\Sigma' \stackrel{def}{=} \Sigma$$

M' has a new tape symbol:

$$\Gamma' \stackrel{def}{=} \Gamma \cup \{\triangleright\}, \text{ where } \triangleright \notin \Gamma$$

We use the same accept and reject states:

$$q'_{accept} \stackrel{def}{=} q_{accept} \quad q'_{reject} \stackrel{def}{=} q_{reject}$$

We will use a bunch of *new* states, exactly $|\Sigma| + 2$ of them, of which q'_0 is our new starting state:

$$Q' \stackrel{def}{=} Q \cup \{q'_0, q_{back}\} \cup \{q_c : c \in \Sigma\}$$

Before we define the new transition function δ', let us see how M' simulates M. M' starts with a string $w \in \Sigma^*$ on the tape. It then shifts w one cell to the right, and writes the new tape symbol $\triangleright$ on the leftmost cell. It then moves its head to the beginning of w and switches to the original state q_0. At this point, the configuration of M' will be $(q_0, \triangleright, w)$. Next, M' will use the original transitions of M:

$$\forall q,q' \in Q, \forall c,c' \in \Gamma, \forall d \in \{L,R\}, \delta'(q,c) \stackrel{def}{=} (q',c',d) \Leftrightarrow \delta(q,c) = (q',c',d)$$

We also have transitions which prevent M' from going left of the end of the tape:

$$\forall q \in Q, \delta'(q,\triangleright) \stackrel{def}{=} (q,\triangleright,R)$$

So now, when M would like to move left of the end of the tape, M' will move the head on to $\triangleright$, and then right with the next transition, simulating the behaviour of M.

We shift the original input as follows:

$$\begin{aligned}
\forall c \in \Sigma, \delta'(q'_0, c) &\stackrel{def}{=} (q_c, \triangleright, R) \\
\delta'(q'_0, \sqcup) &\stackrel{def}{=} (q_0, \triangleright, R) \\
\forall c, c' \in \Sigma, \delta'(q_c, c') &\stackrel{def}{=} (q_{c'}, c, R) \\
\forall c \in \Sigma, \delta'(q_c, \sqcup) &\stackrel{def}{=} (q_{back}, c, L) \\
\forall c \in \Sigma, \delta'(q_{back}, c) &\stackrel{def}{=} (q_{back}, c, L) \\
\delta'(q_{back}, \triangleright) &\stackrel{def}{=} (q_0, \triangleright, R)
\end{aligned}$$

Problem 3

Let $S \stackrel{def}{=} \mathcal{N}^3 = \{(i,j,k) : i,j,k \in \mathcal{N}\}$. Here is an one-to-one (ie, injective) mapping f:

$$\begin{aligned}
f: \quad S &\longrightarrow \mathcal{N} \\
(i,j,k) &\longmapsto 2^i \cdot 3^j \cdot 5^k
\end{aligned}$$

It's easily seen and shown that f is injective. Suppose we're given two triples $(i,j,k) \neq (i',j',k')$, differing in at least one coordinate — without loss of generality, say $i < i'$. Then

$$\frac{f(i,j,k)}{2^i} = 3^j \cdot 5^k$$

is not divisible by 2, while

$$\frac{f(i',j',k')}{2^i} = 2^{i'-i} \cdot 3^{j'} \cdot 5^{k'}$$

is divisible by 2, and thus $f(i,j,k) \neq f(i',j',k')$.

Problem 4

a The set of Turing machines that halt when started with the empty string as input is the same as the set of Turing Machines that halt when started with a blank input tape. The latter (HB) has been shown in lecture notes to be recursively enumerable but not recursive.

b Let $L_2 = \{\langle M, w\rangle | M$ never moves its tape head left on input $w\}$. This language is decidable. Assume for simplicity that the Turing Machine is only allowed to move its head left or right, that is, not to stay put. The argument could be adapted to deal with that case as well. We sketch a Turing Machine M_2 deciding this language.

Given $\langle M, w\rangle$, M_2 simulates M on input w for $|w|$ steps. If M ever moved its tape head left within these $|w|$ steps, M_2 rejects. If M halted, we accept. Otherwise, we know M must be reading a blank symbol to the left of the input string w.

We will then continue the simulation of M for another $|Q| + 1$ steps, that is, for as many steps as there are states in M, plus 1. Again, if M moved its head left within these steps we reject, and if M halted without moving its tape head left, we accept. Otherwise, we know M has been through $|Q| + 1$ configurations in which it read a blank, and since there are only $|Q|$ states in M, we derive that it must have passed the same state twice. But then, nothing prevents M from repeating that loop over and over again, because we know it will only move the tape head right, so it will only read blanks, as in the first loop. Thus we conclude that M will forever move its tape head right, so we accept $\langle M, w\rangle$.

c Let $L_3 = \{\langle M_1, M_2\rangle | L(M_1) = L(M_2)\}$. This problem is not recursively enumerable. We know that $\overline{A_{TM}}$ is not recursively enumerable, so it's enough to show $\overline{A_{TM}} \leq_M L_3$ in order to prove our claim.

Hence we must construct a computable function f such that

$$x \in \overline{A_{TM}} \Leftrightarrow f(x) = \langle M_1, M_2\rangle \in L_3$$

Let M_1 be a Turing Machine which does not accept any string, so that $L(M_1) = \Phi$. If x is not of the form $\langle M, w\rangle$, we know that $x \notin A_{TM}$, so $x \in \overline{A_{TM}}$. We define $f(x) = \langle M_1, M_1\rangle \in L_3$ in order to cover this case.

Now suppose $x = \langle M, w\rangle$. We want to build M_2 such that $L(M_1) = L(M_2)$ iff M does not accept w. Remember M_1 is described above, with $L(M_1) = \Phi$. Here is a description of how M_2 works. On any input y, M_2 discards y, writes w on its tape, and then simulates M on input w, taking the same decision as M, in case M ever halts. Observe that this construction is what we want:

$$\langle M, w\rangle \notin \overline{A_{TM}} \Rightarrow M(w) = \text{yes} \Rightarrow \forall y, M_2(y) = \text{yes} \Rightarrow L(M_2) = \Sigma^* \neq L(M_1) = \Phi \Rightarrow \langle M_1, M_2\rangle \notin L_3$$

Similarly,

$$\langle M, w\rangle \in \overline{A_{TM}} \Rightarrow M(w) \neq \text{yes} \Rightarrow \forall y, M_2(y) \neq \text{yes} \Rightarrow L(M_2) = L(M_1) = \Phi \Rightarrow \langle M_1, M_2\rangle \in L_3$$

By the Church-Turing thesis, f described above is computable, therefore $\overline{A_{TM}} \leq_M L_3$. Since $\overline{A_{TM}}$ is not r.e, L_3 is not r.e.

d This problem was more difficult than anticipated, and therefore it will be marked as extra credit.

The solution depends quite heavily on the particular lexicographic ordering of Turing machines chosen. We will show that sometimes the language is recursive, and other times it is not re.

Fix a lexicographic ordering of all Turing machines, $M_1, M_2, \cdots$. Let the corresponding languages accepted by these machines be $L_1, L_2, \cdots$. Let the language of this problem, with respect to the above ordering be called L_4. (That is, $\langle M, x\rangle$ is in L_4 if and only if M accepts x and if M is the i-th Turing machine in the ordering, then M_j does not accept x, for all $j < i$.)

Let M_k be the first Turing machine in the ordering that accepts all strings. Now define $T_1 = L(M_1)$, $T_2 = L(M_1) \cup L(M_2)$, and so on, so that T_i is the union of the languages accepted by the first i Turing machines, $M_1, \ldots, M_i$.

The first case is when all of the languages $T_1, T_2, \ldots, T_k$ are recursive. In this case, we can show that L_4 is recursive as follows.

Fix Turing machines $M_{T_1}, M_{T_2}, \ldots, M_{T_k}$ that always halt and recognize T_1, T_2, ..., T_k respectively. Given input $\langle M, x\rangle$, we describe an algorithm that always halts and accepts L_4. First, check to see if M is equal to any of $M_1, \ldots, M_k$. If not, then reject $\langle M, x\rangle$. Otherwise, assume without loss of generality that M is the i-th Turing machine in the ordering, $M = M_i$, for some $i \leq k$. Simulate $M_{T_{i-1}}$ on x. If it accepts x, then halt and reject $\langle M, x\rangle$. Otherwise, if it halts and rejects x, then simulate M_{T_i} on x, and if it accepts x, then halt and accept $\langle M, x\rangle$. Otherwise halt and reject x.

This simulation always halts since the Turing machines M_{T_i} always halt. If one of the Turing machines occurring before M_i accepts x, then $M_{T_{i-1}}$ will accept x, and therefore we want to reject $\langle M, x\rangle$. Otherwise, if M_{T_i} accepts x and $M_{T_{i-1}}$ rejects x, then we know that x is in L_i but x is not in any L_j, for $j < i$. Lastly, if M_{T_i} rejects x, then we know that M_i must reject x, so we want to reject $\langle M, x\rangle$.

The second case is when for some $i \leq k$, $T_1, \ldots, T_{i-1}$ are recursive but T_i is not recursive. (Note that this is the only other possible case.) Now we will show that in this case, if L_4 is re, then T_i is recursive, contradicting our assumption that T_i is not recursive. Recall that M_{T_i} is the Turing machine that accepts T_i (but doesn't halt on all inputs).

We will describe an algorithm that always halts and accepts T_i. The algorithm on input x behaves as follows. (We want to decide if one of the first i Turing machines in the original list, $M_1, \ldots, M_i$ accepts x.)

Run the Turing machine accepting L_4 in parallel on $\langle M_1, x\rangle$, $\langle M_2, x\rangle, \ldots \langle M_k, x\rangle$ (by dove-tailing). Since M_k accepts every string, it accepts x, and therefore exactly one of M_1, ... M_k is the lexicographically first machine to accept x. If M_j is the first machine to accept x, and $j \leq i$, then we accept x. Otherwise, if M_j is the first machine to accept x and $j > i$, then we reject x.

This algorithm always halts and we will accept x if and only if x is in T_i. Therefore, we have shown that if L_4 is re, then T_i is recursive, contradicting our assumption that T_i is not recursive. Thus, L_4 is not re, in the case where one of the T_i's is not recursive.

Which of the two is the case with the listing presented in class? We observe that a minimal size (with the encoding presented in lecture) legal Turing Machine has 3 states (start, accept, reject) and 3 tape symbols (0, 1, blank). For any of those, only 3 transitions are critical, the ones from the starting state, reading 0, 1 or blank, respectively. The others, although required by the format, do not alter the language accepted by the Turing Machine. For each of the 3 critical transitions, we have 3 possible next states, 3 possible symbols to write, and 2 possible directions to move the head. That makes for at most $(3 \cdot 3 \cdot 2)^3$ meaningfully distinct Turing Machines at the beginning of our listing. Observe how (at least) one of those machines accepts all strings. There is consensus among the TAs and the prof that probably none of the Turing Machines lexicographically less than the first one which accepts everything is accepting a non-recursive language. This puts us in the first case described above, where L_4 is recursive.

e Let $L_5 \stackrel{def}{=} \{\langle M, x\rangle | M$ is the lexicographically first Turing Machine that accepts the number x within x steps $\}$. L_5 is decidable, essentially because there is a finite number of Turing Machines to simulate, each for a finite amount of steps. Here is a high-level algorithm to decide it:

On input $\langle M, x\rangle$, let $\langle M'\rangle$ be the first valid encoding of a Turing Machine. If $\langle M'\rangle > \langle M\rangle$, reject (neither M nor any Turing Machine less than M accepts x in less than x steps). Simulate M' on input x for up to x steps. If M' accepts and $M' = M$, accept. If M' accepts and $M' \neq M$, reject. If M' does not accept (rejects or didn't stop yet), let $\langle M'\rangle$ be the next valid Turing Machine encoding and go back to testing whether it is greater than $\langle M\rangle$.

By the Church-Turing thesis, there is a Turing Machine implementing the above algorithm, thus deciding L_5.

Remark that this procedure still works when presented with an arbitrary ordering on the Turing Machines (say, given by a bijective function from the natural numbers to the set of valid Turing Machine encodings).

Problem 5

Let us be given a language L which is *accepted* by the Turing Machine M_1, and whose complement, $\overline{L}$, is accepted by M_2. Then, by definition, both L and $\overline{L}$ are recursively enumerable. Our goal is to construct a Turing Machine M *deciding* L, which would show that L is recursive.

So, given an arbitrary string $w \in \{0,1\}^*$, M has to halt on input w, accepting iff $w \in L$. We know that either $w \in L$ or $w \in \overline{L}$, so exactly one of M_1 and M_2 will halt and accept w — we don't know which of them. The idea is to try them both. But we cannot run them one after the other (say M_1 then M_2) because we do not know whether the first one (in the example above, M_1) will ever halt to alllow us to run the second (M_2). So what we have to do is to run them "in parallel". More precisely, we alternatingly simulate one step of the computation of each.

Here is one possible high-level description of a 2-tape M. We start with an arbitrary w on the input tape. We copy w on the second tape, and we then bring both tape heads back on the first symbol on each tape. At this point, we regard the states of M as a cross-prduct of the states of M_1 and of M_2 (remember that we know M_1 and M_2 when we build M), together with an indicator of which one is to be simulated next.

$$Q_M \supset \{(i, q_1, q_2) | q_1 \in Q_{M_1}, q_2 \in Q_{M_2}, i = 1 \text{ or } 2\}$$

The transition function alternates between simulating a step of M_1 on the first tape, and a step of M_2 on the second tape. This would formally look like:

$$\begin{gathered}
\forall q_1, q_1' \in Q_{M_1}, \forall q_2, q_2' \in Q_{M_2}, \forall c_1, c_1', c_2, c_2' \in \Gamma_{M_1} \cup \Gamma_{M_2}, \forall d \in \{L, R, -\}, \\
\delta_M((1, q_1, q_2), c_1, c_2) = ((2, q_1', q_2), c_1', c_2, d, -) \Leftrightarrow \delta_{M_1}(q_1, c_1) = (q_1', c_1', d) \\
\delta_M((2, q_1, q_2), c_1, c_2) = ((1, q_1, q_2'), c_1, c_2', d, -) \Leftrightarrow \delta_{M_2}(q_2, c_2) = (q_2', c_2', d)
\end{gathered}$$

Furthermore, we want to accept w if M_1 accetps it (ie, $w \in L(M_1) = L$), and reject w if M_2 accepts it. Hence,

$$\begin{gathered}
\forall q_1 \in Q_{M_1}, \forall q_2 \in Q_{M_2}, \forall c_1, c_2 \in \Gamma_{M_1} \cup \Gamma_{M_2}, \\
\delta_M((1, q_{accept}, q_2), c_1, c_2) = (q_{accept}, c_1, c_2, -, -) \\
\delta_M((2, q_1, q_{accept}), c_1, c_2) = (q_{reject}, c_1, c_2, -, -)
\end{gathered}$$

By construction, M accepts whatever M_1 accepts, so $L(M) = L(M_1) = L$. Moreover, since on any w either of the two machines accepts, we know that M will always halt. This shows that M *decides* L, therefore L is recursive.