1. **(20 points):** Kozen, Homework 9, problem 4.
Prove that an r.e. set is recursive iff there exists an enumeration machine that enumerates it in increasing order.

Solution: Let Σ be a finite alphabet, and let $A \subseteq \Sigma^*$ be an r.e. set. To order the elements of A, we will say that $x \prec y$ if $|x| < |y|$. If x and y have the same length, we will order them lexographically:

$$\begin{array}{rcll} \epsilon & = & \epsilon & \\ c \cdot x & \prec & d \cdot y, & \text{if } c \prec d \text{ and } |x| = |y| \\ c \cdot x & \succ & d \cdot y, & \text{if } c \succ d \text{ and } |x| = |y| \\ c \cdot x & \prec & d \cdot y, & \text{if } c = d \text{ and } x \prec y \end{array}$$

I'm assuming that there is some ordering of the symbols in Σ – we can always make one up if none was provided.

Now, I'll present the requested proof.

A is recursive $\Rightarrow$ there exists an enumeration machine that enumerates it in increasing order.
Assume that A is recursive. Then, there exists a total TM that accepts A. Let M_A be such a TM. We now construct an enumeration machine, $M_{A,E}$ that generates each string in Σ^* in order, tests each string with M_A, and if M_A accepts, $M_{A,E}$ writes the string on its output tape. Because M_A is total, $M_{A,E}$ never gets stuck looping while testing a string.

A is recursive $\Leftarrow$ there exists an enumeration machine that enumerates it in increasing order.
We consider two cases, A is finite and A is infinite. If A is finite, then A is regular, which means there is a finite automaton that accepts it. We can build a TM, M whose finite state control implements this finite automaton. Upon reaching the end of the input string, M moves to the accepting state, t, if its state is an accepting state of the finite automaton. Otherwise, M rejects.
If A is infinite, then it has no largest element. Let's say we have an enumeration machine, $M_{A,E}$ that enumerates A. We now build a TM M_A that accepts A. M_A has an extra track onto which it copies its input – let y be the input string. It then runs $M_{A,E}$. Each time $M_{A,E}$ outputs a string, u, M_A compares u with y. If $u = y$, M_A accepts. Otherwise, if $u \succ y$, then M_A rejects. Because $M_{A,E}$ outputs strings in increasing order, we know that once it has output a string that follows u in the ordering, it will never output y. Because A is infinite, we know that $M_{A,E}$ will eventually output a string that is greater than y. Thus, M_A is total and accepts A. This proves that A is recursive.

2. **(20 points):** Kozen, Miscellaneous exercise, problem 111.
One of the following sets is r.e. and the other is not. Which is which? Give proof for both.

(a) $\{M \mid L(M)$ contains at least 481 elements$\}$

Solution: This set is r.e.
As in question 1, we can order all strings in Σ^*. Now, we build a machine M' that simulates M for one step on string 0 (i.e. ϵ); then for one step on string 0 and one step on string 1; then for one step on string 0, one step on string 1, and one step on string 2; and so on. Whenever M accepts a string, M' increments a counter and drops the string from the set that it is simulating. If M rejects a string, then M just drops the string from the set that it is enumerating. If the counter for M' reaches 481, then we know that M has accepted 481 different strings, and M' accepts.
To complete the argument, I need to show that M' is guaranteed to terminate if $L(M)$ contains at least 481 elements. If $L(M)$ contains 481 elements, let A_{481} be the 481 *smallest* elements of $L(M)$ according to the ordering we've defined for strings. Let K be the number by the ordering of strings for the largest element of A_{481}. Let N be the maximum number of steps that M takes to accept any string in A_{481}. Machine M' will have simulated M for at least N steps after a total of $((N+K)^2 + (N+K))/2$ simulation steps. Thus, M' will accept after a finite number of steps.
Note that M' loops if M does not accept at least 481 strings. Thus, $L(M)$ is r.e. but not recursive.

(b) $\{M \mid L(M)$ contains at most 481 elements$\}$

Solution: This set is not r.e.

Let $B = \{M \mid L(M)$ contains at most 481 elements$\}$. Let M be an arbitrary TM. If $L(M) = \emptyset$, then $M \in B$. On the other hand, if $L(M) = \Sigma^*$, then $M \notin B$. Thus, B is nonmontone. By the second form of Rice's theorem, we conclude that B is not r.e.

For those that think that was too easy (or aren't sure how to apply Rice's theorem), here's a bit more explanation. First, consider why our approach from part (a) doesn't work here. Consider some machine M that accepts any string w if $|w| > 10^{10^{10^{10^{10^{10^{10^{10^{10^{10}}}}}}}}}$. Clearly, $M \notin B$. On the other hand, we won't figure this out until we've simulated M for roughly

$$\frac{1}{2}\left(10^{10^{10^{10^{10^{10^{10^{10}}}}}}}\right)^2$$

moves – and that's a lot of moves. More to the point, we never know when we've simulated M for enough moves to rule out the possibility that it accepts more strings.

Showing that the construction from part (a) doesn't work, isn't a proof that B isn't r.e. – perhaps we could find some other construction that would show B is r.e. We can take the basic idea of our construction to create a proof. In particular, we can reduce $\sim HP$ to testing membership in B. Let's say that we have some Turing machine M and some string x. We want to know if M loops on x (i.e. is $M\#x \in \sim HP$). We build a TM M' that does the following:

- When run on input y, M' erases y and writes x onto it's tape.
- M' then performs the actions of M.
- If M halts, M' accepts y.

(Does this sound familiar? Take a look at the solution to HW5, Q. 4. I used exactly the same construction there. Once you understand a few reductions, you can apply them to a very wide range of problems.). Note that M' accepts no strings if M loops on input x, and M' accepts an infinite number of strings if M halts on input x.

We now ask if $M' \in B$, and observe that $M' \in B \Leftrightarrow M\#x \in \sim HP$. Thus, we can solve the complement of the halting problem by determining that a machine accepts at most 481 strings. The complement of the halting problem is not r.e., therefore B is not r.e.

At this point, you might object, and say that a very similar argument shows that the halting problem can be reduced to testing membership in

$$\{M \mid L(M) \text{ contains at least 481 elements}\}$$

Indeed it can. The halting problem is r.e., but not recursive. We can give a definite "yes" answer to the question of whether or not M halts on input x, but we can't give a definite "no" in all cases where M does not halt.

3. **(20 points):** Kozen, Miscellaneous exercises, problem 37, parts a, b, i and j.
Which of the following sets are regular and which are not? Give justification.

(a) $\{a^n b^{2m} \mid n \geq 0$ and $m \geq 0\}$

Solution: Regular.

This set is generated by the regular expression $a^*(bb)^*$.

(b) $\{a^n b^m \mid n = 2m\}$

Solution: Not regular.

For the sake of contradiction, assume that the set is regular. Let k be its pumping lemma constant. Now, consider the string $a^{2k}b^k$, and force the demon to pump in the string of a's. Any pumping will change the number of a's while leaving the number of b's unchanged. This produces a string not in the set. This contradicts the assumption that the set is regular.

(c) $\{a^n b^m \mid n \geq m$ and $m \leq 481\}$

Solution: Regular.

We create a DFA with states $\{p_i, q_j, r\}$ for $0 \leq i, j, k \leq 481$. State p_0 is the initial state. Upon reading an a, state p_i transistions to p_{i+1} (for $i \leq 480$); state p_{481} transistions to state p_{481}, and all other states transition to state r. Upon reading a b, states p_i and q_i transistion to q_{i-1} (for $0 < i \leq 481$); and state p_0, q_0, and r transition to r. All states are accepting except for state r.

This machine makes sure that the input is of the form a^*b^*. It also counts the number of a's and makes sure that there are no more b's than a's. If the number of a's is greater than or equal to 481, then it makes sure that there are most 481 b's.

(d) $\{a^n b^m \mid n \geq m$ and $m \geq 481\}$

Solution: Not regular.

For the sake of contradiction, assume that the set is regular. Let k be its pumping lemma constant. Now, consider the string $a^k b^k$, and force the demon to pump in the string of b's. We can pump the string to increase the number of b's while leaving the number of a's unchanged. This produces a string not in the set, contradicting the assumption that the set is regular.

4. **(20 points):** Kozen, Miscellaneous exercises, problem 76.
Consider the set

$$a^*b^*c^* - \{a^n b^n c^n \mid n \geq 0\}$$

the set of all strings of a's followed by b's followed by c's such that the number of a's, b's and c's are not all equal.

(a) Give a CFG for the set, and prove that your grammar is correct.

Solution: The grammar, G,

$$\begin{array}{rcl} S & \rightarrow & U \mid V \\ U & \rightarrow & AXC_0 \mid XBC_0 \\ V & \rightarrow & A_0BY \mid A_0YC \\ A & \rightarrow & a\,A_0 \\ A_0 & \rightarrow & a\,A_0 \mid \epsilon \\ B & \rightarrow & b\,B_0 \\ B_0 & \rightarrow & b\,B_0 \mid \epsilon \\ C & \rightarrow & C\,C_0 \\ C_0 & \rightarrow & c\,C_0 \mid \epsilon \\ X & \rightarrow & a\,X\,b \mid \epsilon \\ Y & \rightarrow & b\,Y\,c \mid \epsilon \end{array}$$

Proof: Let $W = a^*b^*c^* - \{a^n b^n c^n \mid n \geq 0\}$.

$W \subseteq L(G)$: Let $w = a^i b^j c^k \in W$; thus, $i \neq j$ or $j \neq k$. First, consider the case that $i > j$. I'll show that, $S \xrightarrow[G]{1} U \xrightarrow[G]{1} AXC_0 \xrightarrow[G]{*} w$. The first two steps follow directly from the definition of

G. Because $i > j$, we can write w as $a^{i-j}a^jb^jc^k$. It is straightforward to show that:

$$\begin{aligned} A &\xrightarrow[G]{1} a\,A_0 \xrightarrow[G]{i-j-1} a^{i-j}A_0 \xrightarrow[G]{1} a^{i-j} \\ X &\xrightarrow[G]{j} a^jXb^j \xrightarrow[G]{1} a^jb^j \\ \text{and}\quad C_0 &\xrightarrow[G]{k} c^kC_0 \xrightarrow[G]{1} c^k \end{aligned}$$

Thus, $S \xrightarrow[G]{2} AXC_0 \xrightarrow[G]{i+k+3} a^ib^jc^k = w$. This shows that $w \in L(G)$ as required.
The arguments when $i < j$, $j > k$, and $j < k$ are similar.

$W \supseteq L(G)$: Let $w \in L(G)$. First consider the case that $S \xrightarrow[G]{1} U \xrightarrow[G]{1} AXC_0 \xrightarrow[G]{*} w$. It is straightforward to show that for any $\alpha, \beta, \gamma \in \{a, b, c\}^*$,

$$\begin{aligned} A \xrightarrow[G]{i+1} \alpha \quad &\Rightarrow \quad (\alpha = a^i) \wedge (i > 0) \\ X \xrightarrow[G]{j+1} \beta \quad &\Rightarrow \quad (\beta = a^jb^j) \wedge (j \geq 0) \\ \text{and}\quad C_0 \xrightarrow[G]{k+1} \gamma \quad &\Rightarrow \quad (\gamma = c^k) \wedge (k \geq 0) \end{aligned}$$

Thus, $w = a^{i+j}b^jc^k$ with $i + j > j$ (because $i > 0$). This means that $w \in L(a^*b^*c^*)$, and $\neg\exists n.\ w = a^nb^nc^n$. Thus $w \in W$.

The three other cases where $S \xrightarrow[G]{2} XBC_0 \xrightarrow[G]{*} w$, $S \xrightarrow[G]{2} A_0BY \xrightarrow[G]{*} w$, and $S \xrightarrow[G]{2} A_0YC \xrightarrow[G]{*} w$ are similar. Thus, $w \in L(G) \Rightarrow w \in W$ as required.

(b) Give an equivalent PDA.

Solution: Let w be the input string to the PDA. My PDA uses its finite state control to verify that $w \in L(a^*b^*c^*)$. It also makes a non-deterministic choice to determine whether it will show that the number of a's is different than the number of b's or that the number of b's is different than the number of c's.
If it chooses to show that the number of a's is different than the number of b's, it pushes a marker on the stack for each a that it reads. It then pops a marker off the stack for each b that it reads. If it encounters the top-of-stack marker when the next input symbol is a b, it has shown that there are more b's than a's, and moves to a state from which it just checks that the rest of the string is of the form b^*c^*. If there is still at least one marker on the stack the next input symbol is a c or the end of the string has been reached, the machine has shown that there are more a's than b's, and moves to a state from which it checks that the rest of the string is of the form c^*. If the top-of-stack symbol is exposed when it reads the first c, then the machine rejects.
The operation of the machine when it chooses to show that the number of b's is different than the number of c's is similar. It first scans past and a's in the input, pushes a marker for each b it reads, and pops a marker for each c it reads. The details are analagous to those in the previous case.

5. **(20 points):** Kozen, Miscellaneous exercises, problem 106.
Is it decidable, given $M\#y$, whether the Turing machine M ever writes a nonblank symbol on its tape when run with input y? Why or why not?

Solution: Decidable.
Let

$$B \quad = \quad \{M\#y \mid \text{Turing machine } M \text{ writes a nonblank when run on } y\}$$

To make the machine a valid Turing machine, I will assume that when M reads the left tape endmarker, $\vdash$, it writes a $\vdash$ and moves to the right.

I'll show that if a machine writes a nonblank symbol (other than preserving the left endmarker), it must do so within a bounded number of steps. Thus, it is sufficient to simulate the machine for that many steps – if it hasn't written a blank by then, it never will.

Consider a machine that has performed n steps without writing a writing a nonblank. Then, all squares to the left of the tape head must hold the blank symbol (except for the $\vdash$ on the leftmost square). I'll now break the proof into two cases depending on whether or not M reads every symbol of y. I will only consider what we need to do if M only writes blanks. If M ever writes a blank in this procedure, we're done.

M does not read all of y:

Let $n = |y|$ and $k = |Q|$, where Q is the set of states of M. There are n possible positions for M's read write head. If M hasn't written a non-blank, there are n possible tape strings, corresponding to the rightmost position that the head has reached. For each combination of head position and tape string, M can be in any of k possible states. Thus, there are n^2k unique configurations where M has not read all of y.

We can simulate M for n^2k+1 steps. If there are nonblank symbols on the tape at the end of this, then M must have been in the same configuration twice (or else M wrote a nonblank, and we're done). This means that M is looping without writing any nonblank symbols. M will never write a nonblank symbol, and we're done.

M reads all of y:

As described above, we can simulate M for $n^2k + 1$ steps. If M has not written a nonblank and all the tape is all blanks, the only thing that matters is the distance of the head from the left endmarker.

We simulate M for up to k steps. M must visit the same state twice. Let q be such a state. If the second time that M is in state q the head is at the same location or to the right of where it was the first time M was in state q, then M is looping and will continue moving to the right forever. On the other hand, if M has moved overall to the left, then it will continue heading left until it reaches the left endmarker.

M can reach the left endmarker in one of k states. Because the tape is entirely filled with blanks, we can determine what M does in from each of these k configurations. In particular, if M returns to the left endmarker, it must do so within k moves. Thus, we can figure out all possible behaviours within k^2 steps.

Putting these together, we get that after n^2k+1 steps, M has either erased its input or M is in a loop where it never writes a nonblank. If M erases its input, then M is $n + 1$ squares from the left endmarker when it finishes doing so. If M ever returns to the left endmarker, it must do so within $(n + 1)k$ moves (otherwise, we can detect a loop within k moves). If M returns to the left endmarker, the tape is now completely filled with blanks, and we can determine all possible behaviours within k^2 steps.

Thus, we can simulate M on input y for $n^2k + (n + 1)k + k^2 + 1$ steps. If within this number of steps M writes a nonblank, we can answer $M \in B$. Otherwise, we answer $M \notin B$.

6. **(20 points):** Kozen, Miscellaneous exercises, problem 108.
Tell whether the following problems are decidable or undecidable. Give proof.

(a) Given a TM M and a string y, does M every write the symbol $\#$ on input y?

Solution: Undecidable.

Let $B = \{M\#y \mid M \text{ writes } \# \text{ when run on } y\}$. Let M be a Turing machine and x be a string. I'll show how to reduce $M\#x \in HP$ to testing membership in B.

First, I note that given M and x we can check to see if $\#$ is in the tape alphabet of M. If so, we just rename it to another symbol. Thus, I will assume that $\#$ is not in the tape alphabet of M.

Now, I'll construct a machine, M' that when run with input y does the following:

- Erase y and write x on the tape.
- Perform the operation of M.

- If M halts, then write a $\#$ on the tape and accept.

By construction, M never writes a $\#$ on the tape. Thus, M' writes a $\#$ on the tape iff M halts when run with input x. Thus, I've reduced the halting problem to testing whether or not a TM ever writes a $\#$ on its tape. Thus, this problem is undecidable.

Note: saying that this problem is undecidable and saying that B is not recursive are equivalent. I'll note that B is r.e.; thus, this problem is semi-decidable. Simply simulate M with input x. If M ever writes a $\#$, then accept. If M halts without writing a $\#$ reject. Otherwise M loops on input x, and our machine for B will loop when run with $M\#x$.

(b) Given a CFG G, does G generate all strings except ϵ?

Solution: Undecidable.

We saw (Kozen lecture 35) that the question of whether or not $L(G) = \Sigma^*$ is undecidable by the VALCOMPS construction. We note that ϵ is not a valid computational history by the construction used by Kozen (or in class).

Given M and x, let $G_{M,x}$ be a CFG that accepts all strings that are *not* valid computational histories for M run with input x. We can derive a G' such that $L(G') = L(G_{M,x}) = \{\epsilon\}$ – for example, we could use the construction from Kozen chapter 21 (to produce a Chomsky Normal Form grammar) or exploit the fact that CFLs are closed under intersection with regular languages, and $L(\sim\epsilon)$ is regular. The grammar G' generates all strings except ϵ iff M does not halt on input x. Thus, it is undecidable whether or not a CFG generates all strings except ϵ.

(c) Given an LBA M, does M accept a string of even length?

Solution: Undecidable.

In HW6, Q1, we showed that the language emptiness problem for LBAs is undecidable by a reduction from VALCOMPS. In particular, given a TM M and a string x, when can derive an LBA $M_{M,x}$, such that $L(M_{M,x})$ is non-empty iff M halts when run with input x.

We construct an LBA $M'_{M,x}$ that has one more symbol in its input alphabet than $M_{M,x}$. Let $\smile$ be this symbol. We construct $M'_{M,x}$ so that

$$L(M'_{M,x}) \quad = \quad \{x \mid \exists w.(x = w \smile^*) \wedge x \in L(M_{M,x})\}$$

Machine $M'_{M,x}$ first checks to make sure that its input is of the form $\alpha^*_{M,x} \smile^*$ where $\alpha_{M,x}$ matches any symbol in the input alphabet of $M_{M,x}$. (with $M_{M,x}$ modified to treat $\smile$ as the right endmarker). In then replaces the leftmost $\smile$ symbol (if any) with a the right endmarker for $M_{M,x}$ (e.g. $\dashv$), moves its head to the left endmarker, and thereafter operates like machine $M_{M,x}$.

If M halts on input x, then $M'_{M,x}$ accepts strings with an arbitrary number of $\smile$ symbols at the end. Thus, $M'_{M,x}$ accepts strings of even length. On the other hand, if M does not halt on x then $L(M'_{M,x}) = \emptyset$. Hus, $M'_{M,x}$ accepts a string of even length iff M halts on input x.

(d) Give a TM M, are there infinitely many TMs equivalent to M?

Solution: Dedicable.

This is a "trivial" property of the r.e. sets. Given *any* Turing machine, there an an infinite number of equivalent Turing machines.

We say that two Turing machines, M_1 and M_2 are equivalent iff for every possible input string, x, M_1 accepts iff M_2 accepts; M_1 rejects iff M_2 rejects; and M_1 loops iff M_2 loops.

Let $M = (Q, \Sigma, \Gamma, \delta, \vdash, \square, s, t, r)$ be a Turing machine with $Q = \{q_1, q_2, \ldots, q_k\}$. For any $i > 0$, let $M_i = (Q_i, \Sigma, \Gamma, \delta_i, \vdash, \square, s, t, r)$ with $Q_i = Q \cup \{q_{k+1}, q_{k+2}, \ldots, q_{k+i}\}$, and

$$\begin{array}{rcll} \delta_i(q, c) & = & \delta(q, c), & \text{if } q \in Q \\ & = & \delta(q_k, c), & \text{if } q \in Q_i - Q \end{array}$$

Note that there are no transitions into states q_{k+1} and beyond. Thus, they don't affect the computation. Nevertheless, they do result in a different TM. Thus, $L(M_i) = L(M)$, and we can create an infinite

number of TMs this way. We conclude that for any TM M, there are an infinite number of TMs equivalent to M.

You might think that it's somehow "cheating" to add inaccessible states and say that the resulting machine is different from but equivalent to M. I could have added these states so that the new machine would start in state q_{k+i} and work its way down to state q_{k+1} and then transition to state s and act like M after that. If that is still to trivial, I could make M perform some complicated set of actions that end up leaving the tape unchanged, and then transition to state s. While such constructions might help you feel that the machine has a more unique identity, we don't need to resort to such obscurity. Two TMs are identical iff they have the same states, alphabets, transition functions, and special states and symbols. Thus, my simple construction produces distinct TMs.

We could have defined "equivalent" to mean "accept the same set" (thus treating rejection and looping as equivalent). The same construction works.