1) a) Consider a NPDA that operates as follows. It starts by pushing a \$ marker onto the bottom of the stack to indicate the base. The NPDA reads a's from the input and pushes an a onto the stack for each a it encounters. This continues until the first b. As b's are read, the NPDA pops a's from the stack. The point is so that every b is "matched" with an a. When the NPDA reads \$, it starts pushing b's onto the stack for each b read. When a c is encountered, there are 3 possible things that the NPDA knows: 1) if there's an a at the top of the stack, the number of a's on the stack corresponds to the excess of a's over b's $(i-j)$. 2) If there's a b at the top, the number of b's on the stack correspond to the excess of b's over a's $(j-i)$. 3) if the \$ is at the top, the number of a's = the number of b's in the string at this point. If the number of a's on the stack equals the excess of a's over b's, then $i>j$ and we move to an accepting region. If the number of b's on the stack equals the excess of b's over a's, $j \leq i+k$ is met and we accept. If \$ is reached, $i=j$ and we accept. For these cases, all b's must precede c's in the string to accept. If this order is not followed, we reject. To show L_1 is not regular, we will use the pumping lemma. Suppose L_1 is regular. Take string $w=a^p b^{2p} c^p$, where p is pumping length. We can split w/ $w=xyz$, where $|xy| \leq p$, $|y|>0$, and x, y, z are substrings of w/ xyz still belongs to L_1 after pumping. If y spans across the boundary between different types of characters, pumping y will yield a string with characters that are out of order, so the string $\notin L_1$. If y is completely in one type of character, for a and c, pumping down yields a string with an incorrect balance between a's, b's, and c's. If y is entirely in b's, pumping up will also create an invalid string. ∴ the string from pumping y contradicts the $xy^kz \in L_1$ for $k \geq 0$ requirement of the pumping lemma. ∴ L_1 is not regular, but it is context-free as shown by our NPDA model.

b) $L_2 = \{a^i b^j c^k : i=j=k \text{ or } i>1000\}$. If $L' = \{a^i b^j c^k : i=j=k, i \leq 1000\}$ and $L'' = \{a^i b^j c^k : i>1000\}$, $L_2 = L' \cup L''$. Regular languages have closure under union, so if L' and L'' are regular, then so is L_2. For language L', $i,j,k \leq 1000$ since $i \leq 1000$ and $i=j=k$. ∴ L' is a finite language since no character can exceed 1000 and the maximum string length is 3000. Now we have established L' is finite, so we can build a DFA that reads a, b, c and counts their occurrences up to 1000. ∴ Since we can make a DFA with transitions for each string, L' is regular. Next, we must prove $L'' = \{a^i b^j c^k : i>1000\}$ is regular. L'' can be accepted by a DFA that has states counting a's up to 1000. We will construct DFA M as:

→

1) b) - $Q = \{s_0, s_1, s_2 \ldots s_{1000}, s_{1001}, s_{1002}\}$

- $\Sigma = \{a, b, c\}$
- $q_{start} = s_0$ and $q_{accept} = \{s_{1000}, s_{1001}, s_{1002}\}$
- δ: For each a read, we move to the next transition state until >1000 a's are read. If >1000 a's are read, the DFA moves to an accept region. The rest of the string must follow the "b then c" order for the DFA to remain in an accept state. If no b's, then c's follow a's, but all b's should come after a's and before c's.

$\therefore$ we have constructed a DFA that recognizes L'', so L'' is regular. Since L' and L'' are regular and regular languages are closed under union, $L_2 = L' \cup L''$ is regular.

c) We will use the pumping lemma to show whether L_3 is context-free. Let the pumping length for context-free languages be p. We will consider $w = a^q b^q c^q$, where $q \geq \max(p, 1001)$, which is large enough to apply the pumping lemma. $w \in L_3$ since $i=j=k$, so L_3's first condition is met. We can decompose w as $w = uvxyz$, where $|vxy| \leq p$, $|vy| > 0$, and $uv^k xy^k z \in L_3$ for all $k \geq 0$. The substring v and y must consist of symbols from string w. We have two cases for v and y. If v and/or y straddle the boundary between different types of characters, then pumping will result in a string that has characters that are out of order. Specifically, in any pumped version of the string $uv^k xy^k z$, the characters no longer have $a^i b^j c^k$ form. The second case is if v and y are made entirely of one character. If v and y are within a's or b's, pumping will break the $i=j=k$ rule. If v and y are within c's, the i,j,k balance will also be off. Given $q \geq \max(p, 1001)$, if we pump on v and y, we may get a string where the number of a's, b's, or c's > 1000, but also fails to meet $i=j=k$. Pumping results in strings that do not satisfy the language's constraints, so L_3 is not context-free.

2) a) We must show that $co\text{-}L = \{\langle M\rangle$: Turing Machine M has no unreachable states$\}$ is RE to prove whether L is co-RE. We will let $L' = co\text{-}L$. M' will be our recognizer for L' that simulates M on each string $w_1, w_2 \dots$ in parallel, where in the j^{th} round, it simulates M for j steps on $w_1, w_2 \dots w_j$. M' starts by enumerating all strings in L^*, the set of all possible inputs. In each round, M' checks that each state of M has been visited at least once. If at any point all the states of M have been visited, M' will accept the encoding $\langle M\rangle$. If all states of M are reachable, there will be a finite set of strings that when simulated on M, will visit each state. When M' simulates M for enough steps on these strings, all states will be observed, and it will accept. If M has an unreachable state, M' will never visit that state and will simulate M without accepting. $\therefore$ if a state is unreachable, the simulation will run forever. Since we can recognize when all states are reachable, L' is RE and thus L is co-RE.

b) We will reduce E_{TM} to the problem of deciding whether a given TM M has an unreachable state. This reduction will show that if we could decide L, then we could also decide E_{TM}, which is undecidable, so we will have a contradiction. We will construct M' given an instance of $\langle M\rangle \mid L(M') = L(M)$ and M' must visit all states to accept. M' is a copy of M and we will enumerate the states of M (except q_{accept}) as $q_0, q_1, \dots q_n$. The transition to q_{accept} will be replaced by an identical transition to a new state q_{visit}. When in q_{visit}, for each symbol a, M' writes % and transitions to a state $q_{visit+1}$. When all states are visited, M' finally moves to q_{accept}. If w is not accepted by M, M' will not accept w as M' will run endlessly and never reach q_{accept} due to the transitions in the q_{visit} states. Conversely, if w is accepted by M, M' will accept w, but will first write two % symbols and move back and forth over them, visiting each state of M. Once all states are visited, M' enters q_{accept}. Every state gets visited when M accepts w. If $L(M') = 0$, M' has an unreachable state because if M' accepts, it must visit all states of M. $\therefore$ if M' has an unreachable state, M also has an unreachable state. When $L(M') \neq 0$, some string is accepted by M', and every state of M' is reachable. $\therefore$ all states of M are reachable. M' has no unreachable states iff w is accepted by M. $\therefore$ the decision process for L would allow us to decide E_{TM}, which is known to be undecidable. $\Rightarrow\Leftarrow$ Reducing E_{TM} to L shows that L is undecidable because otherwise, E_{TM} would have to be decidable.

3) Proof by contradiction. Assume L_1 and L_2 are recursively separable. $\exists$ a decidable language D where $L_1 \cap D = 0$ and $L_2 \subseteq D$. $\therefore$ there is a TM M that recognizes D. Consider TM $M' = M(\langle M \rangle)$ | we have two possibilities:

1. if M' accepts, $\langle M \rangle \in D$. By definition $\langle M \rangle$ is in L_1. This is a contradiction as $L_1 \cap D = 0$. $\therefore$ M' can't be in both L_1 and D or the intersection of the two would not be empty.
2. if M' rejects, $\langle M \rangle \notin D$. By definition, $\langle M \rangle$ is in L_2. However, $L_2 \subseteq D$, so if $\langle M \rangle \in L_2$, then $\langle M \rangle \in D$. we have a contradiction as $\langle M \rangle \notin D$.

Therefore by contradiction, we have proven L_1 and L_2 are not recursively separable and are thus recursively inseparable.

4) a) Consider the grammar G with $G = \{V, \Sigma, R, S\}$ where V is the finite set of non-terminal symbols, Σ is a finite set of terminal symbols, R is a finite set of production rules | $A \to xB$ for non-terminals $A, B \in V$ and $x \in \Sigma^*$ or $A \to x$ for some non-terminal $A \in V$ and $x \in \Sigma^*$, and $S \in V$ is the start variable. We can prove that $L(G)$ is regular by constructing a NFA that recognizes $L(G)$. We will construct $M = (Q, \Sigma, \delta, q_0, F)$ where Q is all states, Σ is the alphabet, q_0 is the start state and F is the set of accept states. Let δ represent the transition function | $\delta(A, x) = \{B \mid A \to xB \in R\}$. We define this transition function as such: we construct a state for every non-terminal and we create an accept state. For every (A, x) that produces $A \to xB$ where A, B are non-terminals, let $|x| = s$ | we construct $s-1$ states in Q to connect A and B. $\forall$ (A, x) that produces $A \to x$ where A is a non-terminal, then x is a string of terminals. $\therefore$ If $|x| = s$, we can construct $s-1$ states that connect A to an accept state. $\therefore$ we have defined the transition function. Since we can construct a NFA for every language generated by a right-linear CFG, every language generated by a right-linear CFG is regular.

b) We will construct a right-linear CFG for any FA that recognizes the language. First, we will define an NFA $M = (Q, \Sigma, \delta, q_0, F)$ where Q is the finite set of all states, Σ is the alphabet, δ is the transition function $\delta : Q \times \Sigma \to 2^Q$, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accept states. We want to construct a right-linear CFG G | $L(G) = L(M)$. We have $G = (V, \Sigma', R, S)$, where $V = Q$ | every state of M is a non-terminal of G. $\Sigma' = \Sigma$, which is the input alphabet of M. $S = q_0$ as the start symbol corresponds to M's start state. Our productions | R replicates δ will be as follows.
Let $R = \{q \to xr \mid r \in \delta(q, x)\} \cup \{q \to x \mid q \in F\}$. $\forall$ transition x between 2 non-accept states in M, we consider the production $A \to xB$ where A, B are non-accept states in Q and x represents the symbol of transition between states. Similarly, consider the transition between non-accept A and accept state B | we represent this as production $A \to x$ with x as the transition symbol between states.

$\longrightarrow$

4) b) Now we will prove $L(G) = L(M)$. First, if M accepts a string $w = x_1 x_2 \ldots x_n$, then there are states $q_0, q_1 \ldots q_n$ in M | $q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \xrightarrow{x_3} \cdots \xrightarrow{x_n} q_n$. q_n is the accept state. Since we defined the grammar so that each transition $q \xrightarrow{x} r$ in M corresponds to $q \to xr$ in G, G has an equivalent derivation $S \Rightarrow x_1 q_1 \Rightarrow x_1 x_2 q_2 \Rightarrow \ldots \Rightarrow x_1 x_2 \ldots x_n q_n$. Since q_n is an accept state, we apply $q \to r$, which yields $S \Rightarrow x_1 x_2 \ldots x_n$. $\therefore$ w is in $L(G)$ and $L(M) \subseteq L(G)$. Next, if G generates a string $w = a_1 a_2 \ldots a_n$, there is a derivation $S \Rightarrow x_1 q_1 \Rightarrow x_1 x_2 q_2 \Rightarrow \ldots \Rightarrow x_1 x_2 \ldots x_n q_n$ and q_n is the accept state. The grammar was constructed from the automaton, so each production $q_i \to x q_{i+1}$ corresponds to a transition $q_i \xrightarrow{x} q_{i+1}$ in M. We reach q_n, so the automaton accepts w. w is in $L(M)$ so $L(G) \subseteq L(M)$. $\therefore L(M) = L(G)$, which means every regular language is generated by some right-linear CFG.

c) Consider the context-free grammar G with $S \to aT \mid \epsilon$ and $T \to Sb$. We will do induction on n to prove that $\forall n \geq 0$, $a^n b^n$ is in $L(G)$. Assume all strings in L with length $< n$ are derivable. For the base case when $n = 0$, the string is ϵ. This is generated with $S \to \epsilon$. Inductive hypothesis: assume that for some $n \geq 0$, G generates $a^n b^n$. $S \Rightarrow^* a^n b^n$. By production rule $S \to aSb$, a is introduced and b is not introduced yet due to the new S. With our inductive hypothesis, $S \Rightarrow aSb \Rightarrow a(a^n b^n)b = a^{n+1} b^{n+1}$. $\therefore$ G generates all strings in form $a^n b^n$, so $L \subseteq L(G)$. In the other direction, we will use induction on the length of the derivation to show that if G generates w, it is in form $a^n b^n$. Base case for $k=1$: if $S \to \epsilon$, we get ϵ, which is in L. Inductive hypothesis: assume any string derived in $\leq k$ steps is in $a^n b^n$ form. Inductive step: if the derivation starts with $S \to aT$, the next step must be $T \to Sb$, which leads to $S \Rightarrow aT \Rightarrow aSb$. Since $S \Rightarrow^* a^n b^n$ in $< k$ steps, $S \Rightarrow aT \Rightarrow aSb \Rightarrow a(a^n b^n)b = a^{n+1} b^{n+1}$. It follows that if the first step is $T \to Sb$, we know by induction that $T \Rightarrow Sb \Rightarrow aTb \Rightarrow a(aTb)b \Rightarrow aaSbb$. Since $S \Rightarrow^* \epsilon$, we get $aabb$, which is in form $a^n b^n$ also. $\therefore L(G) \subseteq L$. Therefore G exactly generates the language $L = \{a^n b^n \mid n \geq 0\}$.

5) If AT-LEAST-50_L is RE, $\exists$ a TM that recognizes it. The TM M' for AT-LEAST-50_L will simulate the recognizer M for L on each x_i in parallel and accept the input if >50 of the x_i's belong to L. For each $\#x_1\#x_2\ldots\#x_k\#$ for some $k \geq 0$, simulate M for L on each x_i. Since M is a recognizer for L, M will halt and accept iff $x_i \in L$ and may run forever if $x_i \notin L$. M' simulates M on all x's, simultaneously. For each increasing step $S = 1, 2, 3, \ldots$ After simulating M for S steps, count the number of x_i's for which M has accepted. If >50 of the x_i's have been accepted by M, halt and accept the input. This ensures we have seen >50 elements of L. If after enough steps, we have seen >50 acceptances, the machine accepts. Otherwise, if <50 x_i's are in L, the simulation will continue indefinitely and the machine will never accept. Since the recognizer for AT-LEAST-50_L can simulate M for each x_i and halt when it observes >50 of the simulations accept, AT-LEAST-50_L is RE. $\therefore$ if L is RE, AT-LEAST-50_L is RE.