(1) For any string \( x \in \{0, 1\}^* \), let \( m = \#1(x) \), and let \( n_x = (n_1, \ldots, n_{m-1}) \) stand for the numbers of 0s between each 1 in \( x \) and the next 1 in \( x \). Zeroes before the first 1 and after the last 1 in \( x \) do not count, so if \( x \) has at most one 1, then \( n_x \) is the empty vector. For the following conditions \( a, b, c \) let \( L_a, L_b, L_c \) stand for the language of strings meeting the corresponding condition:

(a) Every \( n_i \) in the list \( n_x \) is an even number.

(b) The list \( n_x \) is in nondecreasing order, i.e., \( n_1 \leq n_2 \leq \cdots \leq n_{m-1} \).

(c) Every number in \( n_x \) is different; i.e., for all distinct \( i, j \in \{1, \ldots, m-1\} \), \( n_i \neq n_j \).

For each of \( L_a, L_b, L_c \), say whether it is a regular language or not. If you say regular, give a DFA or regular expression whose correctness is clear. If you say not regular, you must give a proof via the Myhill-Nerode technique.

An element of this problem is that each of these definitions might be thought of as ambiguous when the list \( n_x \) is empty. You must make your interpretation of such cases explicit. Your DFA or regular expression will require such an interpretation anyway, but the nub of the matter is, in the case(s) where you say “not regular,” to make your proof work regardless of the interpretation. (27 points total)

Answer: (a) When the list is empty, it is easy to regard a statement such as “every \( n_i \) in the list...” as true by default. So the empty-list cases \( 0^* \) and \( 0^*10^* \) should count as belonging to \( L_a \). With this said, the only way a string can fail to be in \( L_a \) is if it has two 1s with an odd number of 0s between them. So the complement of \( L_a \) is \( \Sigma^*1(00)^*01\Sigma^* \). Thus the complement is regular, so \( L_a \) is regular. Since the failure condition of an “odd block” of 0s makes the string dead, we can design a DFA whose only rejecting state is its dead state.

For \( L_b \), a list with one block is generally considered to be in nondecreasing order. The empty list is a different story but you can take a cue from how a natural definition of “nondecreasing” would start by saying, “there do not exist two blocks such that...” The empty list furnishes no blocks, so it cannot have a violation.

In any event, \( L_b \) is nonregular, and to show this, we need only care about strings with two blocks: those of the form \( 10^{m_1}10^{m_2}1 \). So we can do the proof as though the language were \( L'_b = \{10^i10^j : i \leq j\} \). Take \( S = 10^*1 \), which is clearly infinite. Let any \( x, y \in S, x \neq y \), be given. Then we can write \( x = 10^m1 \) and \( y = 10^n1 \) where without loss of generality, \( m < n \). Take \( z = 0^m1 \). Then \( xz = 10^m10^m1 \) is in \( L'_b \)—and also in the original \( L_b \)—because “nondecreasing” allows equality. But \( yz = 10^n10^m1 \) is not in \( L'_b \)—nor in the original \( L_b \)—because \( n > m \) and there is no other way of parsing the string into blocks. Thus \( L_b(xz) \neq L_b(yz) \), and since \( x, y \) are arbitrary in the infinite set \( S \), \( L_b \) is not regular by MNT.

For \( L_c \), once again the condition begins with a universal-quantifier word: “Every number in \( n_x \) is different...” So it should be true for the empty list by default. Lists with one element pose a
verbal problem: can the one element be “different”? The issue is finessed, however, by rewriting the definition a little more clearly: “For all pairs \( n_i, n_j \) (being numbers of 0s in distinct blocks of \( x \)), \( n_i \neq n_j \). That wording clearly defaults to true if there are no pairs because the lists has only one or zero elements. So again, \( 0^* \) and \( 0^*10^* \) should be considered to be subsets of the language \( L_c \).

The point is mooted by the same reasoning as for \( L_b \), however: we need only care about strings with two blocks. The same proof (actually not needing “wlog. \( m < n \) this time) works. The only difference is that now \( xz = 10^m10^m1 \) is not in \( L_c \), but \( yz = 10^m10^m1 \) is in \( L_c \) because \( m \neq n \). Thus, \( L_c(xz) \neq L_c(yz) \) anyway, and the rest is as before.

**General comment:** many people did not answer about the empty list or other “edge cases” at all.

(2) Write in pseudocode a decision procedure for the following decision problem:

**INSTANCE:** Two DFAs \( M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1) \) and \( M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2) \).

**QUESTION:** Is \( L(M_1) \cup L(M_2) = \Sigma^* \)?

You may appeal to the DFA minimization algorithm in Debray’s notes as a given but need not reiterate its details. Also say whether your procedure runs in time that is polynomial in the number \( m = |Q_1| + |Q_2| \) of states in the DFAs. (18 pts.)

**Answer:** Make \( M_3 = (Q_3, \Sigma, \delta_3, s_3, F_3) \) by doing the Cartesian product construction with \( \cup \) as the operator. Then \( L(M_3) = L(M_1) \cup L(M_2) \), so \( L(M_1) \cup L(M_2) = \Sigma^* \iff L(M_3) = \Sigma^* \), that is, if \( \langle M_3 \rangle \in ALL_{DFA} \). Now to solve whether \( L(M_3) = \Sigma^* \), we could apply the DFA minimization algorithm and accept if and only if it leaves us with a DFA that consists of a single accepting state that always loops to itself. That’s fine, because the minimization algorithm uses (a rudimentary form of) “dynamic programming” (DP) in a way that runs in polynomial time. However, it is simpler to do what lecture did and make \( M_4 = (Q_3, \Sigma, \delta_3, s_3, F_4) \) with \( F_4 = Q_3 \setminus F_3 \) as we’ve done before. Then \( L(M_3) = \Sigma^* \iff L(M_4) = \emptyset \). We can decide \( L(M_4) = \emptyset \) by doing breadth-first search from \( s_3 \) and saying “yes” if no accepting state (if any) is found to be reachable from \( s \), on any sequence of characters. BFS is “gentler” than DP and it is easier to see that it runs in polynomial time.

That is to say, both DP and BFS run in polynomial time in the size of \( M_3 \) or \( M_4 \), which is what they are given. The size is basically the size of \( Q_3 \). By the Cartesian product construction, \( |Q_3| \leq |Q_1| \times |Q_2| \times (|Q_1| + |Q_2|)^2 = m^2 \). So the overall runtime is “polynomial in \( m^2 \)” which can be re-stated as “polynomial in \( m \)” (General comment: many seemed to think it was also given that \( L(M_1) \) and \( L(M_2) \) are disjoint.)

(3) Let \( A \) be a c.e. language and let \( B \) be a co-c.e. language. Prove that their difference \( A \setminus B \) is always a c.e. language. Give an example, however, where \( A \) is decidable and yet the difference \( A \setminus B \) is still undecidable. (15 pts., for 60 on the assignment)

**Answer:** Because \( B \) is co-c.e., its complement \( \breve{B} \) is c.e. Now \( A \setminus B = A \cap \breve{B} \), so the language \( C \) we want is the intersection of two c.e. languages. We can take a TM \( M_A \) such that \( L(M_A) = A \) and a TM \( M_B \) such that \( L(M_B) = \breve{B} \). We can then build a TM \( M_C \) that on any input \( x \) first runs \( M_A(x) \) and \( M_B(x) \) and accepts if and when both machines accept. Then \( L(M_C) = C \) and thus \( C \) is c.e. (Because this is intersection, it does not matter if we run \( M_A(x) \) before starting on \( M_B(x) \) or do the two simulations in parallel. If the operation were union, however, the series option could fail if \( M_A(x) \) did not halt.)

For the second question, take \( A = \Sigma^* \) (which is “trivially” decidable) and \( B = D_{TM} \) (or \( B = \) any other co-c.e. language that is not c.e.). Then \( A \setminus B \) equals the complement of \( D_{TM} \) (i.e., it equals \( K_{TM} \), which is likewise undecidable.)