1. (5 Points) Construct a PDA with the language 
\[ \{0^m1^n \mid m \geq n \geq 0\} \].

**Proof.** Informally: While we see zeroes in the input, put zero onto the stack (if the first symbol is 1 then reject). Once we see a 1, start popping zeros off the stack. If we see a zero in the input after the first 1, or run out of zeros to pop, reject. Otherwise accept once we reach the end of the input.

2. (10 Points) Let \( L^\pi \) denote the set of strings that can be obtained by permuting a string in the language \( L \). For example, \( \{\epsilon, 112, 123\}^\pi = \{\epsilon, 112, 121, 211, 123, 132, 213, 231, 312, 321\} \). Prove that regular languages are not closed under the \( \pi \) operation.

**Proof.** Consider the language \( L = (01)^* \). \( L \) is regular, but \( L^\pi = \{x \in \{0, 1\}^* : \) the number of zeros in \( x \) is equal to the number of ones in \( x\} \), which we know is not regular.

3. (5 Points) Let \( M_1 \) and \( M_2 \) be two Turing machines. Circle around the correct words (separated by or) so that the following statement is correct, and explain your answer:

The language of the following TM is \( (L(M_1) \cap L(M_2) \) or \( L(M_1) \cup L(M_2)) \):

On input \( w \):

- Step 1: Run \( M_1 \) on \( w \). If \( M_1 \) accepts or rejects \( w \), then accept or reject, and terminate.
- Step 2: Run \( M_2 \) on \( w \). If \( M_2 \) accepts or rejects \( w \), then accept or reject.

**Proof.** If we want to ensure that \( M \) recognizes \( L(M_1) \cap L(M_2) \), the only combination that makes sense is rejects, reject, and accepts, accept. This way, we run \( M_2 \) on \( w \) if and only if \( M_1 \) accepts \( w \), and we accept \( w \) if and only if \( M_2 \) accepts \( w \). So \( L(M) = L(M_1) \cap L(M_2) \).

Comment: Note that we cannot accept \( L(M_1) \cup L(M_2) \). Indeed we have to wait until \( M_1 \) halts on its input before we can move on to running \( M_2 \) on \( w \). If \( M_1 \) loops forever on \( w \) without rejecting, then we'll never be able to run \( M_2 \) on \( w \), even if \( w \in L(M_2) \). So right away, we can rule out \( L(M_1) \cup L(M_2) \).

4. (10 Points) Let \( A, B, \) and \( C \) be regular languages. Is the following language context-free?

\[ \{xyz \mid x \in A, y \in B, z \in C, |x| = |y| = |z|\} \].

**Proof.** The answer is no. Let \( A, C = 0^*, B = 1^* \). Then the language in question becomes \( \{0^n1^n0^n : n \geq 0\} \). We have already seen that this language is not context-free. However, we will repeat the proof. Suppose that this language is context-free. Then it satisfies the pumping lemma for some constant \( p \). Take the string \( s = 0^p1^p0^p \) where \( q > p \). Then there is no decomposition of \( s \) into \( uvxyz \) such that \( \forall i \geq 0 \ uv^ixy^iz \) is of the form \( 0^p1^p0^p \) (since \( |vxy| < p < q \), we can increase the length of at most two of the three components of the string belonging to \( A, B, C \) when we pump \( 0^q1^q0^q \)).
5. (10 Points) Recall that

\[ A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts } w \} \].

Prove that the Kolmogorov function \( K(x) \) is computable relative to \( A_{TM} \).

Proof. As a refresher: given a string \( x \), \( K(x) \) is the minimum length of a string of the form \( \langle M, w \rangle \) such that on input \( w \), \( M \) writes \( x \) to the tape and halts. Note that we can always modify such a Turning Machine so that when it halts it accepts (not reject). Now, we construct a Turing machine to compute \( K(x) \) as follows: on input \( x \), iterating from \( n = 1 \) to infinity, query \( A_{TM} \) with each string \( s \in \Sigma^* \) of length \( n \), where \( \Sigma \) is the alphabet of the Turing machine. If \( A_{TM} \) says that \( s \) is of the form \( \langle M, w \rangle \) where \( M \) accepts \( w \), then run \( M \) on \( w \) and see if it writes \( x \) to the tape. If it does, then erase everything and write the value of \( n \) to the tape, otherwise, continue through iteration.

Since we will eventually write every string in \( \Sigma^* \), we will eventually write down a description of every possible \( \langle M, w \rangle \), and since the Turing machine which does nothing and leaves \( w \) on the tape halts with \( x \) on the tape when given input \( x \), we will always be guaranteed to compute some value for \( K(x) \). Further, since we test strings in order of length, the first string that results in \( x \) being written to the tape is the smallest string with this property, so this Turing machine computes the correct value.

6. Rigorously establish the decidability or undecidability of the following languages:

(a) (10 Points)

\[ L = \{ 0^n | \text{the decimal expansion of } \pi = 3.14 \ldots \text{ contains } n \text{ (or more) consecutive } 0\text{'s} \} \].

Decidable.

Proof. Let \( M \in \bar{N} = \mathbb{N} \cup \{ \infty \} \) be the maximal number of consecutive zeros in \( \pi \). Then if \( M < \infty \), \( L = \{ 0^n : n < M \} \). Else, \( L = 0^* \). In both cases, \( L \) is regular and hence decidable.

(b) (10 Points)

\[ L = \{ \langle G \rangle \mid G \text{ is a context-free grammar and } L(G) \text{ is finite} \} \].

Decidable.

Proof. Rewrite \( G \) into Chomsky normal form (this is possible to do algorithmically). Now observe that \( G \) is infinite if and only if for some symbol \( A \) in \( G \), there is some derivation of \( A \) that produces \( A \). Thus, to decide this language we need only go through the finitely many symbols in \( G \) and determine the set of variables:

\[ A_x = \{ x : x \text{ is reachable from } A \} \]

This can be done by an algorithm which marks \( A \), then marks all of the symbols reachable by following one rule from \( A \), then all of the symbols generated by following one rule from any marked symbol, and continuing until we don’t mark a new symbol. If \( A \in A_x \) for any \( A \) in our CFG, then \( G \) is infinite. Thus, if \( \forall A \in A_x \), then \( A \) is finite, and we can determine this in a finite amount of time.

So we construct a Turing machine \( T \) which on input \( \langle G \rangle \), converts \( G \) to Chomsky normal form, then for each symbol in \( G \) performs the algorithm described above. If we find an \( A \mid A \in A_x \), then reject. Otherwise, once all symbols have been tested, accept.
(c) (10 Points)

\[ L = \{ \langle D \rangle \mid D \text{ is a DFA and } L(D) \text{ is not recognized by a DFA with fewer states} \}. \]

Decidable.

**Proof.** Let the number of states in \( D \) be \( k \). Then there are finitely many DFAs with the same alphabet as \( D \) with fewer than \( k \) states. We can then let \( T \) be a Turing machine which on input \( D \) of length \( k \), constructs each DFA on fewer than \( k \) states and compares \( D \) with this DFA and decides if their languages are equal. Then if it ever constructs a DFA \( E \) such that \( L(E) = L(D) \), it rejects. Otherwise, after testing each possible DFA and not rejecting, it accepts.

It is possible to decide whether the language of two DFA’s is the same for the following reason. Let \( D_1 \) and \( D_2 \) be two DFA’s, and note that \( L(D_1) = L(D_2) \) if and only if \((L(D_1) \setminus L(D_2)) \cup (L(D_2) \setminus L(D_1)) = \emptyset\). We know how to construct a DFA \( D_3 \) whose language is \((L(D_1) \setminus L(D_2)) \cup (L(D_2) \setminus L(D_1))\). Now our task is to see whether \( D(L_3) = \). This can be checked easily by running a depth first search on the DFA to see if we can reach any accept states from the start state. \( \square \)

(d) (10 Points)

\[ L = \{ \langle M, w \rangle \mid M \text{ is a TM and on input } w \text{ it does not change anything on the tape} \}. \]

Decidable.

**Proof.** Let \( q \) denote the number of states in \( M \), and \( n = |w| \). We note that if a Turing machine never writes to its tape, then we have one of 3 cases:

i. it eventually accepts or rejects its input and halts without having written anything;

ii. it loops forever over some bounded section of the tape;

iii. it loops in a way that takes it left or right along the tape forever.

Consider the section of the tape which consists of the input \( w \), plus the \( q \) cells to the right of it. The key point to solve this problem is to notice that if \( M \) ever leaves this region without having written anything, then we are in case (iii). Indeed consider the first time that the head reaches the cell \(|w|+q+1\). Let us say this happened at time \( t_1 \), and let \( t_0 \) be the last time that the head was on the input part of the tape. Obviously \( t_1 - t_0 > q \), and moreover note that during the time-interval \([t_0 + 1, t_1]\) (whose length is larger than \( q \)) the head was always reading blanks from the tape. Since the machine has only \( q \) states, a state is visited more than once, moving the tape further to the right. Since the head is now in the blank section, this will show that the head will keep moving right and that state will be revisited again and again. This results in an infinite loop.

On the other hand if the tape head never leaves the first \( n+q \) cells of the tape, we can treat it as if it has bounded memory. We will call the combination of head location and state a **configuration** of this Turing machine. Then if \( M \) never writes to the tape, we can reach at most \( q(n+q) \) distinct configurations before we have to loop back to something we’ve seen before (case (ii)). So if we run \( M \) on \( w \) for more than \( q(n+q) \) steps and find it does not write to the tape, it will never write to the tape.

With these observations, we now construct a Turing machine \( R \) which decides \( L \). On input \( \langle M, w \rangle \), simulate \( M \) on \( w \) for \( q(n+q) \) steps (or until it halts, if that occurs first). If \( M \) writes to the tape, reject. If \( M \) does not write to the tape, accept. \( \square \)
(e) (10 Points)

\[ L = \{ \langle M, w \rangle \mid M \text{ is a TM and on input } w \text{ it modifies at most 10 memory cells} \}. \]

Here these 10 memory cells can be anywhere on the tape.

Decidable.

**Proof.** Using the result from Part (d), we let \( T \) be a Turing machine which decides \( L_d = \{ \langle M, w \rangle \mid M \text{ is a TM and on input } w \text{ it does not change anything on the tape} \} \), and define a Turing machine \( S \) which decides \( L \) as follows: on input \( \langle M, w \rangle \), run \( T \) on \( \langle M, w \rangle \). If \( T \) accepts, then accept. Else, simulate \( M \) on \( w \) until it writes a symbol to the tape. Let \( w_1 \) be the current state of the tape after \( M \) has written to it. Then run \( T \) on \( \langle M, w_1 \rangle \). Repeat this process until we either accept, or reach \( w_{11} \), in which case we reject. \( \square \)

7. (10 Points) Let \( A \) be any language in \( P \) over the alphabet \( \{0, 1\} \). Prove that

\[ L_A = \{ 1^n \mid n \in \mathbb{N}, A \cap \{0, 1\}^n \neq \emptyset \} \]

is in \( NP \).

**Proof.** To prove that a language \( L \) is in \( NP \), we must show that there exists a Turing machine \( M \) that can decide \( L \) in nondeterministic polynomial time. Suppose that algorithm \( B \) decides the language \( A \) in polynomial time. For the language \( L_A \), define a Turing machine \( M \) as follows: on input \( 1^n \), non-deterministically choose a string in \( \{0, 1\}^n \) and run \( B \) on this string. If \( B \) accepts, then accept. \( \square \)