Problem 1 [40 points, 2 points per statement]: (Proven) True, (Proven) False, or Open—Circle One

(a) O : \( P = \text{PSPACE} \).

(b) F : \( P = \text{EXP} \).

(c) O (Note: all answers were marked as correct): If \( P = \text{NP} \) then \( \text{NP} = \text{EXP} \).

(d) F : No consistent axiomatic theory can prove its own inconsistency.

(e) T : \( \{ \langle M \rangle : M^A(\cdot) \text{ accepts for some oracle } A \} \) is Turing-equivalent to \( \text{HALT} \).

(f) F : The set of context-free languages is uncountable.

(g) T : The set of non-context-free languages is uncountable.

(h) O : If there exists a polynomial-time factoring algorithm, then \( P = \text{NP} \).

(i) T : \( K(x\#y) \leq K(x) + K(y) + O(\log n) \) for all \( n \)-bit strings \( x, y \).

(j) F : \( K(x\#y) \geq K(x) + K(y) - O(\log n) \) for all \( n \)-bit strings \( x, y \).

(k) T : For all \( n \), there exists a circuit consisting only of NOR gates, which accepts an \( n \)-bit string \( x \) if and only if \( K(x) \geq n/2 \).

(l) T : \( 3^{3\sqrt{\ln n}} = 2^{o(n)} \).

(m) F : \( 2^n = \Theta(3^n) \).

(n) F : All computable languages are mapping-reducible to each other.

(o) T : There exists a language \( L \) such that \( L \not\leq_T \text{HALT} \) and \( \text{HALT} \not\leq_T L \).

(p) T : If there’s an NDFA for \( L \), then there’s an NDFA for \( \overline{L} \).

(q) F : If there’s an NPDA (Nondeterministic Pushdown Automaton) for \( L \), then there’s an NPDA for \( \overline{L} \).

(r) T : If there’s a DPDA (Deterministic Pushdown Automaton) for \( L \), then there’s a DPDA for \( \overline{L} \).

(s) T : \( \{ \langle M \rangle, y : y \text{ encodes a ZF proof that } M(\cdot) \text{ halts} \} \) is decidable.

(t) T : There exists a Turing machine \( M \) such that (i) \( M(x) \) halts for every input \( x \), but (ii) there is no proof in ZF that \( M(x) \) halts for every \( x \).

---

1Here and in the next two problems, \( K(x) \) is the Kolmogorov complexity of \( x \) (the length of the shortest program that outputs \( x \)), and \# denotes concatenation of two strings.

2You can assume that the constants 0 and 1 are available to the circuit for free.

3You can assume for this question that ZF is sound.
Problem 2 [20 points + 5 extra credit points]: Regular and Context-Free Languages

For parts (a)-(d), you do not need to prove your construction’s correctness. Also, given an $n$-bit string $x = x_1 \cdots x_n$, a prefix of $x$ just means one of the $n$ substrings $x_1, x_1x_2, x_1x_2x_3, \ldots, x_1 \cdots x_n$.

(a) [5 points] Let $L = \{ x \in \{0,1\}^* : \text{no prefix of } x \text{ has both an odd number of } 0\text{'s and an odd number of } 1\text{'s} \}$. Show that $L$ is regular, by giving a DFA that recognizes it.

(b) [5 points] Give a regular expression for the language $L$ from part (a).

$$(00|11)^* (0|1)\cdot$$

(c) [5 points] Let $L' = \{ x \in \{0,1\}^* : \text{no prefix of } x \text{ has more } 1\text{'s than } 0\text{'s} \}$. Show that $L'$ is context-free, by giving a (deterministic or nondeterministic) pushdown automaton that recognizes it. You can either draw the PDA, or just describe it verbally.

(d) [5 points] Give a context-free grammar for the language $L'$ from part (c).

$$(\text{Start}) \to S$$

$S \to \epsilon | 0S1 | S0 | SS$$

(e) [5 points, extra credit] Show that the language $L'$ from part (c) is not regular. [Hint: While it’s possible to do this with the pumping lemma, we’d suggest directly using the pigeonhole principle.]

Let $M$ be a DFA for $L'$. By pigeonhole, there must exist $n < n'$ such that $M$ ends up in the same state after scanning $0^n$ or $0^{n'}$. Hence $M$ behaves identically on the two inputs $0^n1^n$ and $0^{n'}1^n$. But $0^n1^n \in L'$ while $0^n1^n \notin L'$, which is a contradiction.
Problem 3 [20 points + 20 extra credit points]: Magic Advice Strings

Given a language \( L \subseteq \{0, 1\}^n \), let’s call \( L \) computable with \( f(n) \) bits of advice if there exists a Turing machine \( M \), together with a collection of \( f(n) \)-bit “magic advice strings” \( a_n \in \{0, 1\}^{f(n)} \) (one for each positive integer \( n \)), such that \( M(x, a_n) \) correctly decides whether \( x \in L \) for every \( n \) and every \( n \)-bit input \( x \in \{0, 1\}^n \).

To put it differently: when given an input \( x \) of length \( n \), the Turing machine \( M \) receives a “hint from an all-knowing teacher,” which can be anything whatsoever that \( M \) wants (even solutions to instances of the halting problem, etc). The only restrictions are that (i) the hint can only be an all-knowing teacher,” which can be anything whatsoever that \( M \) wants (even solutions to instances of the halting problem, etc). The only restrictions are that (i) the hint can only be \( f(n) \) bits long, and (ii) crucially, the hint can only depend on the input length \( |x| = n \), not on any other information about the input \( x \).

To illustrate the definition, it’s easy to see that a language is computable with 0 bits of advice if and only if it’s computable.

(a) [5 points] Explain why every language \( L \) is computable with \( 2^n \) bits of advice.

With \( 2^n \) bits of advice, we can simply encode whether or not \( x \in L \) for all \( x \in \{0, 1\}^n \) (i.e. all \( n \)-bit inputs).

(b) [5 points] Show that there exists a language \( L \) that’s computable with 1 bit of advice, but not computable with 0 bits of advice. (You can assume the unsolvability of the halting problem. Unary encoding might be helpful.)

Let \( L = \{0^n : M_n() \text{ halts}\} \). If \( L \) were decidable, then clearly HALT would be (albeit exponentially slower). On the other hand, \( L \) is computable with a single advice bit that’s 1 if \( M_n() \) halts and 0 if not.

(c) [10 points] Recall that \( \text{BLANKHALT} = \{\langle P \rangle : P() \text{ halts}\} \). Show that \( \text{BLANKHALT} \) is computable with \( n = |\langle P \rangle| \) bits of advice. (In other words: what’s an \( n \)-bit string \( a_n \) that, if you knew it, would let you solve the halting problem on all programs of length \( n \)?)

We can let \( a_n \) be the number of \( n \)-bit programs \( P \) such that \( P() \) halts. Then given an \( n \)-bit \( \langle Q \rangle \), we can decide whether \( Q() \) halts by running all \( n \)-bit programs in parallel until \( a_n \) of them have halted. If \( Q \) is among these \( a_n \), then \( \langle Q \rangle \in \text{BLANKHALT} \); otherwise not.

Alternatively, we can let \( a_n \) be an \( n \)-bit program \( P \) such that \( P() \) runs for the maximum number of steps, among all \( n \)-bit programs that halt (in other words, such that \( P() \) runs for \( BB(n) \) steps). Then given \( \langle Q \rangle \), we can decide whether \( \langle Q \rangle \in \text{BLANKHALT} \) by seeing whether \( Q() \) halts in at most the number of steps that \( P() \) takes to halt.

(d) [10 points, extra credit] Is every language \( L \) computable with \( n \) bits of advice? Why or why not?

No, not every \( L \) is so computable. To see this, we use a Shannon-like counting argument. Given a Turing machine \( M \), consider the language (if any) decided by \( M(x, a_n) \), as we vary over all collections of \( n \)-bit strings \( \{a_n\}_{n \geq 1} \). There are \( 2^n \) possible \( n \)-bit strings \( a_n \). On the other hand, there are \( 2^{2^n} \) possible Boolean functions \( f : \{0, 1\}^n \rightarrow \{0, 1\} \) – in other words, possibilities \( L_n = L \cap \{0, 1\}^n \) for \( L \) on the \( n \)-bit strings. Since \( 2^{2^n} > 2^n \), it follows that for every \( n \), there must be some way to set \( L_n \) such that \( M(x, a_n) \) faults to decide \( L_n \) regardless of how we set \( a_n \). But this means we can form \( L = L_1 \cup L_2 \cup L_3 \cup \ldots \) by “diagonalizing” against all Turing machines – i.e., causing \( M_n \) to fail on \( L_n \), where \( M_1, M_2, M_3, \ldots \) is an enumeration of TMs. Then the resulting \( L \) will not be computable by any
TM $M$ using $n$ bits of advice.

(e) [10 points, extra credit] Is $\text{BLANKHALT}$ computable with $n/2$ bits of advice? Why or why not? [Hint: Try using Kolmogorov complexity to argue by contradiction!]

No, it isn’t. Suppose by contradiction that $\text{BLANKHALT}$ were computable with $n/2$ bits of advice. Note that, by padding out shorter programs with garbage, we could certainly use the advice $a_n$ to decide $\text{BLANKHALT}$ on programs $\langle P \rangle$ shorter than $n$ bits, as well as programs that were $n$ bits exactly. Therefore, we could write a program that produced a listing of all programs $\langle P \rangle$ such that $P()$ halts and $|\langle P \rangle| \leq n$. Then, we could run each of those programs to completion, and find the shortest one whose output is $x$, in order to compute $K(x)$ given any $n$-bit string $x$. This, in turn, would let us write a program that searched for and output the lexicographically first $n$-bit string $z$ such that $K(z) \geq 0.99n$ (such a $z$ must exist, by pigeonhole). Further, the size of this program would be at most $n/2$ (the length of $a_n$), plus $O(\log n)$ (the number of bits needed to specify $n$), plus $O(1)$ for control. Therefore $K(z) \leq n/2 + O(\log n)$. But that contradicts $K(z) \geq 0.99n$. Therefore $\text{BLANKHALT}$ can’t be computed with $n/2$ advice bits.
**Problem 4 [20 points]: Write Complexity**

In class, we saw time and space complexity, but one can also define other complexity measures. As an example, say a Turing machine *writes to the tape* when it replaces the symbol on the current square by a different symbol (leaving the symbol unchanged doesn’t count as “writing”). Then let $\text{WRITE}(f(n))$ be the class of languages $L \subseteq \{0,1\}^*$ that are decidable by a one-tape Turing machine $M$, with an alphabet of any constant size, such that $M$ writes to the tape at most $O(f(n))$ times when given an $n$-bit string $x$ as its input. Also, by analogy to $\text{P}$ and $\text{PSPACE}$, let $\text{PWRITE} = \bigcup_k \text{WRITE}(n^k)$.

(a) [5 points] Explain why $\text{TIME}(f(n)) \subseteq \text{WRITE}(f(n))$ for any $f$ (and thus, $\text{P} \subseteq \text{PWRITE}$).

Writing to the tape certainly takes one time step! Thus, any Turing machine that writes $f(n)$ times must run for at least $f(n)$ steps. In other words, if you use at most $f(n)$ steps then you write at most $f(n)$ times. So $\text{TIME}(f(n)) \subseteq \text{WRITE}(f(n))$.

(b) [10 points] Show that $\text{PWRITE} \subseteq \text{P}$. (Combined with part (a), this implies that $\text{P} = \text{PWRITE}$.)

Suppose a Turing machine writes to tape at most $f(n)$ times. Then I claim that the machine can run for at most $O(f(n)(n+f(n)))$ steps (where the constant in the big-O depends on the number of states in the machine) without getting into an infinite loop. To see this, note that between two write steps, the machine is just reading and moving left and right on the tape. But how many different configurations can the machine get into, without either writing to tape or re-entering a previous configuration? Clearly, the answer is at most the “tape width” (i.e. the maximum number of squares the machine can visit at this point), times the number $|\langle M \rangle|$ of internal states of the machine. But the latter is just a constant (independent of $n$), while the former is at most $2T|\langle M \rangle| + n$ after the $T$th write step (why? Because the machine can’t venture more than $|\langle M \rangle|$ tape squares out into a ”wilderness of zeroes” without writing anything, without getting stuck in an infinite loop.) So the maximum number of steps is

$$\sum_{T=1}^{f(n)} |\langle M \rangle|(n + 2T|\langle M \rangle|) \leq n \cdot f(n) \cdot |\langle M \rangle| + f(n)^2|\langle M \rangle| = O(f(n)(n+f(n))).$$

Setting $f(n) = \text{poly}(n)$, it follows that $\text{PWRITE} \subseteq \text{P}$, and hence $\text{P} = \text{PWRITE}$.

(c) [5 points] Recall that in class, we discussed the language $\text{PALINDROME} = \{x \in \{0,1\}^* : x \text{ is a palindrome}\}$, and asserted (without proof) that deciding $\text{PALINDROME}$ requires $\Omega(n^2)$ time for a one-tape Turing machine. Show that $\text{PALINDROME} \in \text{WRITE}(n)$—thereby proving that, for one-tape Turing machines, the class $\text{WRITE}(f(n))$ is not equal to $\text{TIME}(f(n))$ in general.

To put $\text{PALINDROME}$ in $\text{WRITE}(n)$, we can simply use the Turing machine from class – the one that shuttles back and forth on the input tape, checking that the appropriate pairs of symbols match (e.g., in 0110, the two zeros match and the two ones match) and successively overwriting them with #’s if they do. This machine uses $\Theta(n^2)$ time (due to the need to shuttle back and forth across the input), but it only writes $n$ times – once for each input bit – in the worst case. Since $\text{PALINDROME} \notin \text{TIME}(n)$, it follows that $\text{WRITE}(n)$ strictly contains $\text{TIME}(n)$. 
