A. If N uses space $s(n)$, then there is an equivalent deterministic Turing machine M that uses space $\mathcal{O}(s(n)^2)$, by Savitch's theorem. Then M must use time $2^{\mathcal{O}(s(n)^2)}$, by one of the time-space relationships proven in class.

B. A suitable string is the nested sum $({}^p1[+1)]^p$, where the [] are metacharacters expressing a grouping, not literal characters to appear in the string. For example, if $p = 5$ then the string is $(((((1 + 1) + 1) + 1) + 1) + 1)$. This is a valid Python expression; it evaluates to $p + 1$.

C. We define a Turing machine D that, on input x, outputs $\langle K(x) \rangle$ as follows. This D loops over all bit strings y, in lexicographic order. For each y:

1. Check that y is of the form $\langle M, w\rangle$, where M is a Turing machine and w is an input for M. If not, then abort this y and proceed to the next y.
2. Run H on $y = \langle M, w\rangle$. If H rejects, then abort this y and proceed to the next y.
3. Run M on w.
4. If the output of M is x, then set the tape to $\langle |y| \rangle$ and accept. Otherwise, proceed to the next y.

First, notice that each step within D's loop halts. Second, when D is dealing with a particular y, D will output $\langle |y| \rangle$ if and only if y is a description of x. Third, recall that there is a constant c such that $K(x) \leq |x| + c$ for all x. Therefore, D will find a description of x among the strings y of length at most $|x| + c$. Finally, because the y are tried in lexicographic order, the $\langle |y| \rangle$ that D outputs must be the length of the *minimal* description of x, and thus $\langle K(x) \rangle$.

D1. Briefly, finite $\subseteq$ reg $\subseteq$ context-free $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ EXPTIME $\subseteq$ decid $\subseteq$ recog.
D2. ACC_{TM} is recognizable but not decidable. So is HALT_{TM}. So is $\overline{\text{EMPTY}_{\text{TM}}}$.
D3. Any of the NP-complete problems is suitable: SAT, 3SAT, $\neq$SAT, CLIQUE, MAX-CUT.

E. The answer is P. To prove so, I need to prove two statements: that P is big enough, and that no smaller class is big enough.

First, I argue that if $A \leq_p B$ and B is context-free, then $A \in$ P. Let F be a $\mathcal{O}(n^k)$-time reduction from A to B. All context-free languages are in P, as we've proven in class. So there exists a $\mathcal{O}(n^\ell)$-time decider M for B. Then an algorithm for deciding A is: Given w, compute $F(w)$. Then run M on $F(w)$, and output whatever M outputs. This algorithm runs in time $\mathcal{O}(n^k + \mathcal{O}(|F(w)|^\ell) = \mathcal{O}(n^k) + \mathcal{O}((n^k)^\ell) = \mathcal{O}(n^{k\ell})$. Thus $A \in$ P.

Second, I argue that if A is any language in P, then there exists a context-free B such that $A \leq_p B$. Let $A \in$ P, and let $B = \{1\}$. Let M be a polynomial-time decider for A. Define a reduction F from A to B as follows. On input w, F runs M on w. If M accepts, then F outputs

1. If M rejects, then F outputs 0. This construction shows that $A \leq_p B$. Finally, B is finite, and hence regular, and hence context-free.

F. [Although justification is not required, I give it anyway, for educational value.]
F1. TRUE. [Time complexity is defined only for Turing machines that halt on all inputs. If a Turing machine didn't halt on an input of length n, then its time complexity would be infinite.]
F2. TRUE. [We proved in class that implementing a multi-tape Turing machine on a one-tape Turing machine causes at most a quadratic blowup in running time. So, if M runs in time $\mathcal{O}(n^k)$, then there is a one-tape version that runs in time $\mathcal{O}(n^{2k})$, which is still polynomial.]
F3. FALSE. [We can conclude that B is NP-hard. But we do not know that B is in NP.]
F4. FALSE. [Our proofs of Savitch's theorem and the fact that TQBF is PSPACE-complete used divide-and-conquer, but our Cook-Levin proof did not.]
F5. FALSE. [Every non-empty A in P is PSPACE-complete. But $A = \emptyset$ and $A = \Sigma^*$ are not.]
F6. TRUE. [We mentioned this in class. If there are recognizers for A and $\bar{A}$, then we can run them in parallel to build a decider for A. Once we have a decider for A, we can "negate" it to get a decider for $\bar{A}$.]

G1. TQBF is the set of all fully quantified Boolean formulas that are true. A nontrivial example is

$$\forall x \; \exists y \; ((\exists z \; y \wedge z) \wedge (x \vee y)).$$

A Boolean formula is a formula consisting of variables operated on by and ($\wedge$), or ($\vee$), and not ($\bar{\ }$), and quantified by existential ($\exists$) and universal ($\forall$) quantifiers. The variables can take on the values TRUE and FALSE. The formula is fully quantified if every variable appears inside a quantifier. A fully quantified Boolean formula is either true or false; its truth does not depend on a truth value assignment. The formula above is true because, no matter whether x is TRUE or FALSE, a value of TRUE for y and TRUE for z makes $y \wedge z$ and $x \vee y$ both true.
G2. TQBF is important to computer science because it is PSPACE-complete. PSPACE is the set of computational problems that can be solved using a "reasonable" amount of memory. TQBF is one of these problems, which is not remarkable. What's remarkable is that every such problem can be reduced to TQBF in a "reasonable" amount of time. That is, if we had a time-efficient solution to TQBF, then we would have a time-efficient solution to a huge class of problems. This huge class contains, for example, the integer factoring and discrete logarithm problems, on which all of modern cryptography relies.