1. **(SOLO)** Consider the following computational problem. Given as input a natural number \( N \) in binary (base 2), output \( \lceil N^{1/251} \rceil \) in unary (base 1). Do one of the following:

(i) Give a \( O(n^k) \) time algorithm, where \( n \) is the input length and \( k \) is some constant, to solve this problem. You do not have to argue that your algorithm is correct, but you should argue why the running time is as advertised.

**OR**

(ii) Prove that this problem cannot be computed in polynomial time. (Here, you are allowed to assume that exponentials grow faster than polynomials.)

2. **(GROUP)** Consider the following computational problem. On input a positive integer \( C \), output a rational number \( y \) such that \( e - \frac{1}{C} \leq y \leq e + \frac{1}{C} \). Here \( e \) denotes the base of natural logarithms, \( e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots \), and “outputting a rational number” means outputting its numerator and denominator (written, as usual, in binary).

Determine whether this problem can be computed in worst-case polynomial-time, i.e. \( O(n^k) \) time for some constant \( k \), where \( n \) denotes the number of bits in the binary representation of the input. If you think the problem can be solved in polynomial time, give an algorithm in pseudocode, explain briefly why it gives the correct answer, and argue carefully why the running time is polynomial. If you think the problem cannot be solved in polynomial time, then provide a proof.

3. **(GROUP)** Describe an algorithm that takes as input an \( n \)-digit number in base 10, and outputs the corresponding number in base 2. For full credit, you must present an algorithm with running time better than \( \Theta(n^2) \).

4. **(GROUP)** Let’s play a game. You have 10 seconds to write the biggest number you possibly can. What would you write? Would you write, 999999999... with as many 9’s you can write in 10 seconds? Probably not, you are more sophisticated than that. You might think of writing \( 9^{9^9} \), since you know, exponentials grow super fast. (It is probably better to write \( 7^{7^7} \) since it is faster to write a 7 than a 9.) And that would indeed be an incomprehensibly big number. A rough estimate for the number of subatomic particles in the observable universe is about \( 9^{100} \).

Is exponentiation the fastest growth rate? Certainly not. If multiplication is repeated addition, and exponentiation is repeated multiplication, we can define tetration as repeated exponentiation. The notation is \( ^n x \), which denotes \( x \) raised to itself \( n \) times, e.g., \( ^3 x = x^x^x \). We can of course keep going, and define \( x \) pentated to \( n \) as \( x \) tetrated to itself \( n \) times. And define \( x \) sextated to \( n \) as \( x \) pentated to itself \( n \) times, and so on. The Ackermann function allows us to go beyond all of this. (The Ackermann function actually does show up in mathematics and computer science!) Define \( A(1) = 1 + 1 \), \( A(2) = 2 \times 2 \), \( A(3) = 3^3 \), \( A(4) = 4 \) tetrated to the 4, \( A(5) \) as 5 pentated to the 5, and so on. Then we can name tremendously HIYUUUGE numbers using the Ackermann function, e.g. \( A(999999999) \) (or even \( A(999999999)^{A(999999999)} \)).
Cool, but surely this is not the end. Let’s try to go past exponentials, the Ackermann function, and any other system for naming huge numbers. What if we specified a number by saying “The biggest number that you can specify using a thousand English words or fewer.” But wait. Couldn’t we just then write “Two times the biggest number that you can specify using a thousand English words or fewer.” to specify an even bigger number? Something doesn’t seem to be right. Perhaps it is because the notion of “being specified in English” is not precisely defined. No worries. You are taking 15-251, so you know better. Let’s specify numbers with Turing machines, which we know is a programming language with a very precise definition.

So far we have talked about using Turing machines to decide languages; i.e., solve decision problems. We can also talk about Turing machines solving general computational problems. Suppose $M$ is a decider Turing machine with input alphabet $\Sigma$ and tape alphabet $\Gamma$. We define the output of $M$ on input $x \in \Sigma^*$ as follows: Suppose the final configuration of $M(x)$ is $uqv$, where $u,v \in \Gamma^*$ and $q$ is either the accept or reject state of $M$. Then the output of $M(x)$ is defined to be the longest prefix of $v$ that is in $\Sigma^*$. (Note that it doesn’t matter whether $M(x)$ accepts or rejects.) In this way, every decider TM $M$ can be thought of as computing a function $f_M : \Sigma^* \rightarrow \Sigma^*$. A computational problem $g : \Sigma^* \rightarrow \Sigma^*$ is said to be computable if it is $f_M$ for some decider TM $M$. If $\Sigma = \{0,1\}$ and we think of strings in $\Sigma^*$ as natural numbers (according to binary representation), then this also gives us a definition for when a function $g : \mathbb{N} \rightarrow \mathbb{N}$ is computable. Functions like $g(n) = n^2$, $g(n) = 2^n$, and even the Ackermann function, are all computable.

Fix the input alphabet $\Sigma$ to be $\{0,1\}$ and the tape alphabet $\Gamma$ to be $\{0,1,\square\}$. For an integer $n \geq 2$, let $B(n)$ denote the maximum number of steps that an $n$-state TM can take when run on input $\epsilon$. Here, the maximum is over all $n$-state TMs that halt on input $\epsilon$. Prove that $B(n)$ grows faster than any computable function. More precisely, show that there is no computable function $g : \mathbb{N} \rightarrow \mathbb{N}$ (in the sense of the above definition) such that $g(n) > B(n)$ for all $n$.

So in the game of naming largest numbers, you can now crush everyone using $B(n)$. But for fun, think about whether you can come up with functions that even dwarf the growth rate of $B(n)$.

5. (GROUP) Determine (with proof) whether the following languages are decidable or not.

(a) Is $A = \{\langle M \rangle : M$ is a TM that never goes left on input $\langle M \rangle \}$ decidable?

(b) Call a TM $M$ crazy if running $M$ with input $\langle M \rangle$ makes the TM access at most $251^{251}$ distinct cells of the tape. Is $B = \{\langle M \rangle : M$ is a crazy TM\} decidable?

(c) For a TM $M$, we say that a state $q$ is bad if $M(x)$ never enters $q$ for any input $x$. Is $C = \{\langle M \rangle : M$ is a TM containing a bad state\} decidable?