1. Let $f$ be a non-decreasing, positive integer-valued function over the positive integers. Prove that if $f(n+1) \in O(f(n))$, then there is a $c$ so that $f(n) \in O(2^cn)$. Is the converse always true? Prove it or give a counter-example.

First, if $f(n+1) \in O(f(n))$, by definition of order, there are constants $n_0 > 0, c > 0$ so that for all $n > n_0$, $f(n+1) \leq cf(n)$. (Note this is similar to the recurrence: $T(n) = cT(n-1)$, and the proof that $f$ is exponential is just solving this recurrence. Without loss of generality, assume $c > 1$. We'll prove by induction that for all $n \geq n_0$, $f(n) \leq c^{n-n_0}f(n_0)$. The claim is true for $n = n_0$, since the two sides both evaluate to $f(n_0)$. For the induction step, assuming $f(n) \leq c^{n-n_0}f(n_0)$, for some $n \geq n_0$, we have $f(n+1) \leq cf(n) \leq cc^{n-n_0}f(n_0) = c^{n+1-n_0}f(n_0)$ as claimed. So the claim holds inductively.

Let $d = \log c$, so $c = 2^d$, and let $D = f(n_0)$. Then the above expression becomes $f(n) \leq 2^{d(n-n_0)}D \in O(2^{dn})$. So the order of the function is bounded by an exponential in a linear function as desired.

The converse isn’t always true. We give a counter-example Let $f(n) = 2^{2\log n}$. Then $f(n) \leq 2^{2\log n+1} = 2^{2n}$, so is upper bounded by an exponential. But when $n = 2^j$ is a power of 2, $f(n) = 2^n$ and $f(n+1) = 2^{2\log n+1} = 2^{2n} = 2^n f(n)$. If we had $f(n+1) \leq cf(n)$ for any $n \geq n_0$, we get a contradiction by picking $n$ to be any power of 2 greater than $n_0$ and greater than $\log c$. So $f(n+1)$ is not $O(f(n))$ although $f$ is a non-decreasing integer valued function that is exponentially bounded.

2. Let $L_N$ be the language of binary strings whose number of 1’s is NOT divisible by $N$. Prove that, for infinitely many $N$’s, the number of states needed by a non-deterministic finite automaton recognizing $L_N$ is strictly smaller than that of the smallest deterministic finite automaton doing so.

We first show that, for any $N \geq 2$, $L_N$ requires at least $N$ states to recognize by a deterministic finite automaton. (It is easy to see that there is also a matching upper bound, but it is not needed for this problem.) Consider the inputs $w_i = 1^i$ for $i = \ldots, N - 1$. If in any DFA deciding $L_N$, we had $w_i$ go to the same state as $w_j$ for $i < j$, then $w_i1^{N-j} = 1^i1^{N-j}$ would be in the same state as $w_j1^{N-j} = 1^N$. But the first string has a number of 1’s between 1 and $N - 1$, and so should be accepted, while the second has exactly $N$ 1’s and so should be rejected. Thus, any such DFA needs at least as many states as there are $w_i$’s, or at least $N$ states.

On the other hand, let $N \geq 6$ be of the form $N = 2q$ for $q$ an odd number (equivalently, $N$ is 2 mod 4). An integer $m$ is divisible by $N$ when it is even and divisible by $q$, so it is not divisible by $N$ exactly when it is either odd or not divisible by $q$. We can make a 2 state deterministic finite automaton which accepts only those with an odd number of 1’s, and a $q$ state automaton which accepts those strings with a number of 1’s not divisible by $q$. (where the states are the number of 1’s mod $q$, and reading a 1 takes you to the next state modulo $q$, and the only rejecting state is 0 which is also the starting state). Then we can make an NFA with an additional start state and...
epsilon moves to the start states of these two automata. So the total number of states in the NFA is $q + 3$. When $q \geq 4$, this is less than the $N = 2q$ states we showed were needed for a DFA.

3. Say that you are on a street, and know that the store you are looking for is on the same street, but not how far away or which direction it is. You know it is at least 100 meters away, because you can see 100 meters each way. You will know the store when you get to it. How can you find the store and be sure to walk at most $cd$ total distance, where $c$ is a fixed constant that will come from your solution and $d$ is the distance to the store?

I’ll be a bit lazy and show that there is an algorithm that achieves $c = 16$, although better values are possible. The algorithm is as follows: Call the two directions of the street North and South. Start $D$ at 200 meters. Repeat until you get to the store: Walk $D$ North. Walk $2D$ South. Walk $D$ North (arriving at the start point). Double $D$ and repeat.

Each iteration, we walk $4D$ total distance. Since the distance doubles each iteration, the total distance in this and all previous iterations is at most $4D(1 + 1/2 + 1/4 + 1/8 + 1/16... \leq 8D$ since the series converges to 2. Finally, let $d$ be the actual distance to the store. $d \geq D/2$ in the first iteration, since then $D = 200$ and we are told $d \geq 100$. In all subsequent iterations, since we walked the current value of $D/2$ both North and South in the previous iteration, and did not find the store, we must always have $d \geq D/2$. Thus, when $D$ becomes larger than $d$ for the first time, it will be at most $2d$, and we will find the store. Since the total distance we walked is at most $8D$ and $D \leq 2d$, we walked at most $16d$ total.