1. **Countability Basics**

   (a) Is $f : \mathbb{N} \to \mathbb{N}$, defined by $f(n) = n^2$ an injection (one-to-one)? Briefly justify.

   (b) Is $f : \mathbb{R} \to \mathbb{R}$, defined by $f(x) = x^3 + 1$ a surjection (onto)? Briefly justify.

   Solution:

   (a) Yes. One way to illustrate is by drawing the one-to-one mapping from n to n^2. More formally we can show that the preimage is unique by showing that $m \neq n \implies f(m) \neq f(n)$.
   We'll do proof by contraposition. $f(m) = f(n) \implies m = n$.

   $$f(m) = f(n) \implies m^2 = n^2 \implies m^2 - n^2 = 0 \implies (m-n)(m+n) = 0 \implies m = \pm n$$

   Since n can't be negative, we have an injection.

   (b) Yes. For any value of y, there always exists a corresponding input x. If $y = x^3 + 1$, we know that $x = \sqrt[3]{y-1}$. Thus for any value of y, there exists this value of x which maps to it.

2. **Count It!**

   For each of the following collections, determine and briefly explain whether it is finite, countably infinite (like the natural numbers), or uncountably infinite (like the reals):

   (a) The integers which divide 8.

   (b) The integers which 8 divides.

   (c) The functions from $\mathbb{N}$ to $\mathbb{N}$.

   (d) Computer programs that halt.

   (e) Computer programs that always correctly tell if a program halts or not.

   (f) Numbers that are the roots of nonzero polynomials with integer coefficients.

   Solution:

   (a) Finite. They are $\{-8, -4, -2, -1, 1, 2, 4, 8\}$.

   (b) Countably infinite. We know that there exists a bijective function $f : \mathbb{N} \to \mathbb{Z}$. Then function $g(n) = 8f(n)$ is a bijective mapping from $\mathbb{N}$ to integers which 8 divides.

(c) Uncountably infinite. We use the Cantor's Diagonalization Proof:

Let $\mathscr{F}$ be the set of all functions from $\mathbb{N}$ to $\mathbb{N}$. We can represent a function $f \in \mathscr{F}$ as an infinite sequence $(f(0), f(1), \cdots)$, where the i-th element is $f(i)$. Suppose towards a contradiction that there is a bijection from $\mathbb{N}$ to $\mathscr{F}$:

$$\begin{aligned}
0 &\longleftrightarrow (f_0(0), f_0(1), f_0(2), f_0(3), \cdots) \\
1 &\longleftrightarrow (f_1(0), f_1(1), f_1(2), f_1(3), \cdots) \\
2 &\longleftrightarrow (f_2(0), f_2(1), f_2(2), f_2(3), \cdots) \\
3 &\longleftrightarrow (f_3(0), f_3(1), f_3(2), f_3(3), \cdots) \\
&\vdots
\end{aligned}$$

Consider the function $g : \mathbb{N} \to \mathbb{N}$ where $g(i) = f_i(i) + 1$ for $i \in \mathbb{N}$. We claim that the function g is not in our finite list of functions. Suppose for contradiction that it did, and that it was the n-th function $f_n(\cdot)$ in the list, i.e., $g(\cdot) = f_n(\cdot)$. However, $f_n(\cdot)$ and $g(\cdot)$ differ in the n-th number, i.e. $f_n(n) \neq g(n)$, because by our construction $g(n) = f_n(n) + 1$ (Contradiction!).

(d) Countably infinite. The total number of programs is countably infinite, since each can be viewed as a string of characters (so for example if we assume each character is one of the 256 possible values, then each program can be viewed as number in base 257, and we know these numbers are countably infinite. So the number of halting programs, which is a subset of all programs, can be either finite or countably finite. But there are an infinite number of halting programs, for example for each number i the program that just prints i is different for each i. So the total number of halting programs is countably infinite.

(e) Finite. There is no such program because the halting problem cannot be solved.

(f) Countably infinite. Polynomials with integer coefficients themselves are countably infinite. So let us list all polynomials with integer coefficients as $P_1, P_2, \ldots$. We can label each root by a pair (i, j) which means take the polynomial P_i and take its j-th root (we can have an arbitrary ordering on the roots of each polynomial). This means that the roots of these polynomials can be mapped in an injective manner to $\mathbb{N} \times \mathbb{N}$ which we know is countably infinite. So this set is either finite or countably infinite. But every natural number n is in this set (it is the root of $x - n$). So this set is countably infinite.

3. **Countable and Uncountable**

(a) Give a bijection from the real number interval $(1, \infty)$ to the real number interval $(0, 1)$. (Notice the intervals are open.)

(b) Given an $n \times n$ matrix A where the diagonal consist of alternating 1's and 0's starting from 1, $A[0,0] = 1$, describe a n length vector from $\{0,1\}^n$ that is not equal to a row in the matrix. (Hint: the all ones vector or the all zeros vector of length n could each be rows in the matrix.)

Solution:

(a) $f(x) = 1/x$

(b) The row consisting of alternating 1's and 0's starting with 0. That is, 010101....

This is supposed to recall the diagonalization idea of constructing something not in a list by making it different along the diagonal.

4. **Computability**

(a) The problem of determining whether a program halts in time 2^{n^2} on an input of size n is undecidable. (True or False.)

(b) There is no computer program DEAD which takes a program P, an input, x, and a line number, n, and determines whether the nth line of code is executed when the program P is run on the input x. (True or False.)

Solution:

(a) False. You can simulate a program for 2^{n^2} steps and see if it halts.

The concept is that you can run a program for any fixed *given* amount of time to see what it does. The problem of undecidability comes in when you don't have a bound on the time.

(b) True.

We implement HALT which takes a program P and an input x using DEAD as follows. We take the input P and modify it so that each exit or return statement jumps to a particular new line. Call the resulting program P'. We then hand that program to DEAD along with the input x and the number of the new line. If the original program halts than DEAD would return true, and if not DEAD would return false.

This contradicts the fact that the program HALT does not exist, so DEAD does not exist.

There are two ways to show undecidability. Use your program as a subroutine to solve a problem we know is undecidable or to do a "diagonalization" proof like we did for HALT. The former is natural for computer programmers and flows from the fact that you are given P as text! Therefore you can look at it and make modifications. This is what the solution above does.