Countability Practice

(a). Yes, they have the same cardinality.

$$f(x) = \frac{1}{x} - 1$$

Suppose f(x) = f(y)

$$\frac{1}{x} - 1 = \frac{1}{y} - 1$$
$$x = y$$

Hence f is one to one.

Take any $y \in (0, \infty)$, let $x \in (0, 1), x = 1/(1 + y)$. Then,

$$f(x) = \frac{1}{x} - 1$$

$$f(x) = \frac{1}{\frac{1}{1+y}} - 1$$

$$f(x) = y$$

So f maps x to y, hence onto.

(b). Yes, it is countable. We can enumerate the strings in such a way that each string appears exactly once in the list. Firstly, list all strings of length 1 in lexicographic order, then all strings of length 2, and then length 3, and so on. Since each step, there are only finite number of strings of a particular length, any string of finite length appears in the list, and it is clear that each string appears exactly once in this list.

(c). The answer do not change, the strings are still countable. We can encode the strings into ternary strings. Suppose we have a string: S = $a_5a_2a_7a_4a6$. Corresponding to each of the characters in this string, we can write its index as a binary string: (101, 10, 111, 100, 110). Moreover, we can construct a ternary string where "2" is inserted as a separator between each binary string. Thus we can map the string S to a ternary string: 101210211121002110. This map is injective, since the original string S can be uniquely recovered from this ternary string. Thus we have an injective map to $\{0, 1, 2\}^*$. Since the $\{0, 1, 2\}^*$ is countable, hence the set of all strings with finite length is also countable.

Fixed Points

We can prove this by reducing from the Halting Problem. Suppose we had some function FixedPoint(F) that solved the fixed-point problem. We can define TestHalt(F,x) as follows:

```
def TestHalt(F, x):
    def F_prime(y):
        F(x)
        return y
    return FixedPoint(F_prime)
```

If $F(x)$ halt, then $F_prime(y)$ return y, and $FixedPoint(F_prime)$ will return true. If $F(x)$ does not halt, then $F_prime(y)$ does not return y, hence $FixedPoint(F_prime)$ return false. Thus TestHalt works! This is contradiction, we conclude.

The Complexity Hierarchy

(a). *enumerator:*

```
for i = 1 to \infty:
    foreach program M where len(M) < l:
        foreach input x where len(x) < l:
            simulate M(x) for at most l steps
                if M(x) has halted, then print out (M,x)
```

(b). *recognizer:*

```
run M(x)
return Yes
```

(c). The desired output for input(M,x) is No if M halts on x, and Yes if M does not halt on x.

(d). Suppose there exists a recognizer Q for the Halting Problem, also a recognizer Q' for the complement. We can define TestHalt(P,x):

```
Alternate between simulating one step of Q(P,x) and Q'(P, x)
if Q(P, x) returns Yes, return Yes
if Q'(P, x) returns Yes, return No
```

We can guarantee that one of Q and Q' will evetually return Yes. Hence the TestHalt will eventually print out Yes if P(x) halts, and No otherwise. However, the TestHalt is not computable, so we have a contradiction. Therefore the Halting Problem cannot be in coRE.

Build Up Error

In the *Inductive Step*, the (n+1)-vertex graph can not only be obtained from a n-vertex graph with minimum 1 degree, but also some graph with 0 degree.

Connectivity

(a). For any two non-adjacent vertices u and v in a graph with n vertices. When deg u + deg v >= n - 1, assume there is no vertex w connected to both u and v, then there only leaves n - 2 vertices for u and v to connect, hence deg u + deg v <= n - 2. We get a contradiction, so there must be at least one vertex w connected to u and v simutaneously, hence the graph is connected.

(b). There are only two non-adjacent vertices u and v, which satisfies the condition deg u + deg v = 0 = 2 - 2 >= n - 2. But the graph is not connected.

(c). For any two non-adjacent vertices u and v, deg u + deg v >= n/2 + n/2 = n > n - 1, by part(a), hence the graph is connected.

(d). There are exactly two vertices with odd degree. Assume they are not connected, then they are in two components, and each component has only one vertex with odd degree and therefore an odd sum of degree. But a connected component can only has an even sum of degree, which is contradict. Hence, we conclude.

Triangular Faces

We can prove by contradiction. Assume there exists one face which is incident on more than three edges. Choose an arbitrary vertex on the face and name it v_0, then with clockwise from v_0, there is v_1, v_2, v_3... Since the face has at least 4 edges, so v_0 and v_2 do not have an edge. If we add an edge between v_0 and v_2, the edge does not cross any other edges and the graph is still planar. Thus the number of edges is $e+1 = 3v-5$, which is contradicted to the theorem that a planar graph can have at most 3v - 6 edges.

Binary Tree

(a). Suppose $u \in L$. Before removing r, u had degree 1 or 3. If u had degree 1, then after removing r, u is a single vertex, and so a binary tree of height 0. If u had degree 3, then after removing r, u has degree 2, and all other vertices in L have degree 1 or 3. Thus, L is a new binary tree with root u.

Formally, we define $\Omega(L)$ and $\Omega(R)$ to be the set of leaves in L and R respectively, and d(r, l) as the length of the path from r to some leaf l, then we have

$$h(T) = \max(1 + \max_{l \in \Omega(L)}(d(u, l)), 1 + \max_{l \in \Omega(R)}(d(v, l))$$

$$= 1 + \max(h(L), h(R))$$

(b). Induction on height h:
Base Case: h = 0, there is $2^{0+1} - 1 = 1$ vertex.
Inductive Hypothesis: Assume $\forall k < h, v \leq 2^{k+1} - 1$.
Inductive Step: By part a, since $h(T) = \max(h(L), h(R)) + 1$, so h(L), h(R) < h, and we can apply the Inductive Hypothesis to L and R. Thus, the number of vertices in T is at most $2^{h(L)+1} - 1 + 2^{h(R)+1} - 1 + 1 = 2^{k+1} + 2^{h(L)+1} + 2^{h(R)+1} - 1 \leq 2 * 2^{h(T)} - 1 = 2^{h(T)+1} - 1$

(c). Induction on the number of leaf l:
Base Case: l = 1, there is $2 * 1 - 1 = 1$ vertex.
Inductive Hypothesis: Assume $\forall l < n, v = 2n - 1$
Inductive Step: l = n, let us remove the root r, the tree T will break into two trees L and R. Suppose L has n_1 leaves and v_1 vertices, R has n_2 leaves and v_2 vertices. $v = v_1 + v_2 = 2 * n_1 - 1 + 2 * n_2 - 1 + 1 = 2 * n - 1$

Euclid's Algorithm

(a). Calculation table lists below.

Function Calls	(x,y)	x mod y
1	(527,323)	204
2	(323,204)	119
3	(204,119)	85
4	(119,85)	34
5	(85,34)	17
6	(34,17)	0
7	(17,0)	–

Hence the gcd(527,323) = 17.

(b). Compute the multiplicative inverse of 5 mod 27.

Function Calls	(x,y)	x mod y	x div y
1	(27,5)	2	5
2	(5,2)	1	2
3	(2,1)	0	2
4	(1,0)	–	–

Then the returned values of all recursive calls are:

Function Calls	(d,a,b)	Returned Values
1	–	(1,1,0)
2	(1,1,0)	(1,0,1)
3	(1,0,1)	(1,1,-2)
4	(1,1,-2)	(1,-2,11)

Thus the the multiplicative inverse of 5 mod 27 is 11.

(c). Find x (mod 27).

$$\begin{aligned} 5x + 26 \equiv 3 \mod 27 &\implies 5x \equiv -23 \mod 27 \\ &\implies 5x \equiv 4 \mod 27 \\ &\implies 11 \times 5x \equiv 11 \times 4 \mod 27 \\ &\implies x \equiv 44 \mod 27 \\ &\implies x \equiv 17 \mod 27 \end{aligned}$$

(d). A counterexample is a = 3, b = 6, c = 12. While a has no multiplicative inverse mod c, but we can get a x = 2 for $ax \equiv b \mod 12$.

GCD Proof

Solution:
1. Prove forward.
Assume there is a gcd d that is greater than 1.

$$\begin{aligned} ax &\equiv 1 \mod n \\ ax &= bn + 1 \\ \frac{ax}{d} &= \frac{bn}{d} + \frac{1}{d} \end{aligned}$$

ax/d and bn/d must both be integers, so 1/d is also an integer, but d > 1, thus 1/d is not an integer, contradict.

2. Prove backward.
We run egcd on n and x, the output is $a, b \in \mathbb{Z}$ and $d = gcd(n, x) = 1$, such that

$$\begin{aligned} an + bx &= 1 \\ bx &\equiv 1 \mod n \end{aligned}$$

Thus, x has a multiplicative inverse b.