In [1]:
```
# Short prime finder
prime_list = [2] + [*filter(lambda i:all(i%j for j in range(3,i,2)), range(3,10000,2))]
prime_set = set(prime_list)
```

In [2]:
```
def is_prime(n):
      return n in prime_set
```

In [3]:
```
# determines if a is a quadratic residue of p
def legendre(a, p, d = 2):
    # if p is prime, simply return the
    if is_prime(p):
        symbol = ((a) ** ((p - 1) // d)) % p
        # handles negative -1
        return symbol if symbol < 2 else -1

    # otherwise, return product of prime factors
    product = 1
    factors = fact(p)
    for i in factors:
        product *= (legendre(i) ** factors[i])
    return product
```

In [4]:
```
# Recursive form of gcd
def gcd(a, b):
    return b if a == 0 else gcd(b%a, a)
```

In [5]:
```
## Extended Euclidean Algorithm
def ext_euclid(a, b):
    a, b = sorted((a, b % a))

    # remainders
    r = [b, a]

    # coefficient of b
    s = [1, 0]

    # coefficient of a
    t = [0, 1]

    # compute values until remainder is 0
    i = 1
    while(r[i] != 0):
        q = (r[i - 1] // r[i])
        r.append(r[i - 1] - q * r[i])
        s.append(s[i - 1] - q * s[i])
        t.append(t[i - 1] - q * t[i])
        i += 1

    # return relevant coefficients and remainder
    return t[i - 1] % b, s[i - 1], r[i - 1]
```

In [6]:
```
# cathode ray tu-- sorry... Chinese Remainder Theorem
# x = a_k mod n_k
def crt(a: list, n: list):
    # first, verify lists are of same size
    assert len(a) == len(n)

    # next, verify coprimality and generate product
    N = 1
    for i in n:
        assert (gcd(i, N) == 1)
        N *= i

    # now, add element N / n_i = y_i to each n
    n = [(i, N // i) for i in n]

    # next, add element multiplicative inverse of y_i = z_i to each n
    n = [i + tuple([ext_euclid(*i)[0]]) for i in n]

    # now, return x = sum(a * y * z) and uniqueness factor
    return sum([a[i] * n[i][1] * n[i][2] for i in range(len(n))]) % N, N
```

In [7]:
```
# Prime Factorialization
from collections import defaultdict

# Prime Factoring Algorithm
def fact(n):
    # dictionary with default value
    out = defaultdict(int)

    # fresh new prime list
    primes = prime_list.copy()

    f = primes.pop(0)
    while f <= n:
        if n % f == 0:
            out[f] += 1
            n //= f
        else:
            f = primes.pop(0)
    return dict(out) # [j for k in [([i] * out[i]) for i in out] for j in k]
```

In [8]:
```
# List comprension to find number of coprimes less than n
def naiive_tot(n):
    return len([i for i in range(n) if gcd(n, i) == 1])
```

In [9]:
```
# totient of a power of a prime
def p_tot(p, n):
    return (p - 1) * p ** (n - 1)
```

In [10]:
```
# finds the totient of a number using its factorialization
def smart_tot(n):
    out     = 1
    factors = fact(n)
    for i in factors:
        out *= p_tot(i, factors[i])
    return out
```

2.3

2. Find all integers that satisfy simultaneously:

$x \equiv 2 \pmod 3, x \equiv 3 \pmod 5, x \equiv 5 \pmod 2$

In [11]:
```
congr = [2, 3, 5]
p     = [3, 5, 2]

# find such an integer using chinese remainder theorem
r = crt(congr, p)

# verify results and print
for i in range(len(p)):
    assert r[0] % p[i] == congr[i] % p[i]
    print("{} + {}n = {} mod {}".format(r[0], r[1], congr[i], p[i]))
```

```
23 + 30n = 2 mod 3
23 + 30n = 3 mod 5
23 + 30n = 5 mod 2
```

4. Find all integers that give the remainders $1, 2, 3$ when divided by $3, 4, 5$, respectively.

In [12]:
```
congr = [1, 2, 3]
p     = [3, 4, 5]

# find such an integer using chinese remainder theorem
r = crt(congr, p)

# verify results and print
for i in range(len(p)):
    assert r[0] % p[i] == congr[i] % p[i]
    print("{} + {}n = {} mod {}".format(r[0], r[1], congr[i], p[i]))
```

```
58 + 60n = 1 mod 3
58 + 60n = 2 mod 4
58 + 60n = 3 mod 5
```

9. For what values of n is $\phi(n)$ odd?

$n > 1$ can be expressed as the product of primes, so $n = 2^m p_1^{a_1} \dots p_k^{a_k}$ where a_i is an odd prime.

Therefore, $\phi(n)$ can be expressed as $\phi(2^m)\phi(p_1^{a_1})\dots\phi(p_k^{a_k})$.

$\phi(p) = p^{k-1}(p-1)$ where p is a prime.

$2 \nmid \phi(p^k)$ where p is prime $\iff 2 \nmid p^{k-1}(p-1) \implies p = 2^1$.

Because all prime numbers except for 2 have even totients, $2 \nmid \phi(n) \iff 2 \nmid \phi(2^m)\phi(p_1^{a_1})\dots\phi(p_k^{a_k}) \iff n = 2$ if $n > 1$.

By observation, $\phi(1) = 1$. Also, note that multiplication by a unit does not affect the value of the totient.

Therefore, $\phi(n)$ is odd $\iff n \in -2, -1, 1, 2$.

10. Find the number of positive integers ≤ 3600 that are prime to 3600.

In [13]:
```
n = 3600

# first, factor number
factors = fact(n)

# let's verify that we have the right factors
res = 1
for i in factors: res *= i ** factors[i]
assert res == n
for i in factors:
    print("({}^{})".format(i, factors[i]), end="")
print(" = {}".format(n))

# present proposition
tot = 1
for i in factors:
    print("ϕ({}^{})".format(i, factors[i]), end="")
print(" = ϕ({})".format(n))

# print results of totient of factors
for i in factors:
    tot *= (e_tot := p_tot(i, factors[i]))
    print("({})".format(e_tot), end="")
print(" = {} = ϕ({})".format(tot, n), end="")

# check with naaive approach
if (naiive_tot(n) == tot):
    print("{}✓".format(" " * 5))
else:
    print("{}:(".format(" " * 5))
```

```
(2^4)(3^2)(5^2) = 3600
ϕ(2^4)ϕ(3^2)ϕ(5^2) = ϕ(3600)
(8)(6)(20) = 960 = ϕ(3600)     ✓
```

2.6

3. Solve $f(x) = x^3 + x + 57 \equiv 0(\mod 5^3)$

We will be using Hensel's Lemma.

Note that $f(4) = (4)^3 + (4) + 57 = 125 \equiv 0(\mod 5)$.

Additionally, $f'(4) = 48 + 1 = 49 \equiv 4(\mod 5) \not\equiv 0(\mod 5)$.

By Hensel's Lemma, there exists a unique solution to the equation $f(x) \equiv 0(\mod 5^{1+2})$ and $x \equiv 4(\mod 5^1)$.

$x = 4 - f(4) \cdot a$ where $a \equiv [f'(4)]^{-1} \mod 5^2$.

A valid integer for a is 4.

Therefore, $x = 4 - 4 * 125 = -496$.

Verifying, $(-496)^3 - 496 + 57 = -122024375 \equiv 0(\mod 5^3)$ 😃

In [14]:
```
assert (4 ** 3 + 4 + 57) % 5 == 0
assert 3 * (4) ** 2 + 1 == 49
assert ((-496) ** 3 - 496 + 57) % (5 ** 3) == 0
```

3.2

2. Prove that if p and q are distinct primes of the form $4k+3$, and if $x^2 \equiv p(\mod q))$ has no solution, then $x^2 \equiv q(\mod p)$ has two solutions.

By the principle of quadratic reciprocity, $(\frac{p}{q})(\frac{q}{p}) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}} = -1$ because $\frac{4k+3-1}{2} = 2k+1$ which is odd.

Because $x^2 \equiv p(\mod q)$ as no solutions, $(\frac{p}{q}) = -1$.

$\implies (\frac{q}{p}) = 1 \implies \exists x : (\pm x)^2 \equiv q(\mod p)$.

6. Decide whether $x^2 \equiv 150(\mod 1009)$ is solvable or not.

In [15]:
```
assert (139 ** 2) % 1009 == 150
assert legendre(150, 1009) == 1
```

139 is a solution, so it must be solvable.

Additionally, 150 is a quadratic residue of 1009, so it solvable.

7. Find all primes p such that $x^2 \equiv 13(\mod p)$ has a solution.

First, we can easily confirm that $p = 2$ has the solution $1^2 = 13 \pmod p$.

For larger, odd primes, we know that $(\frac{13}{p})(\frac{p}{13}) = 1$, as 13 is of the form $4n+1$, so $(\frac{13}{p}) = 1 \iff (\frac{p}{13}) = 1$.

We can quickly find all numbers with $(\frac{n}{13})$, $\{1, 3, 4, 9, 10, 12\}$, so all of the solutions are $p \equiv 1, 3, 4, 9, 10, 12 \pmod{13}$ or $p = 2, 13$.

In [16]:
```
# quick verification script!

for i in range(10000):
    # check that all prime residues have anticipated congruences
    if is_prime(i) and legendre(13, i) == 1:
        assert (i in [2]) or (i % 13) in [1, 3, 4, 9, 10, 12]
        continue

    # check that all primes with congruences have already been found
    if is_prime(i) and ((i in [2]) or (i % 13) in [1, 3, 4, 9, 10, 12]):
        assert False
```

10. Of which primes is -2 a quadratic residue?

First, we know that -2 is clearly a quadratic residue of 2, as $0^2 \equiv -2 \mod 2 \equiv 0 \mod 2$.

$(\frac{-2}{p}) = (\frac{-1}{p})(\frac{2}{p})$ by the properties of the Legendre symbol.

$(\frac{-1}{p})(\frac{2}{p}) = 1 \iff (\frac{-1}{p}) = 1$ and $(\frac{2}{p}) = 1$ or $(\frac{-1}{p}) = -1$ and $(\frac{2}{p}) = -1$.

$(\frac{2}{p}) = 1 \iff p \equiv 1$ or $7 \pmod 8$ (This is a consequence of quadratic reciprocity explained in this document on page 10 https://www.math.brown.edu/~jhs/Frint4thChapter21.pdf (https://www.math.brown.edu/~jhs/Frint4thChapter21.pdf)).

$(\frac{-1}{p}) = 1 \iff p \equiv 1 \pmod 4$. (see the aforementioned document)

Therefore, -2 is a quadratic residue of p if p is of the form $1 \pmod 8$.

Now, we need to account for when $(\frac{2}{p}) = -1$ and $(\frac{-1}{p}) = -1$.

Similarly, we can infer that $(\frac{2}{p}) = -1 \iff p \equiv 3$ or $5 \pmod 8$ and $(\frac{-1}{p}) = -1 \iff p \equiv 3 \pmod 4$, so -2 is a quadratic residue of p if p is of the form $3 \pmod 8$.

Now that we have expressed all of the possibilities for $(\frac{-1}{p})(\frac{2}{p}) = 1$, we can definitively say that -2 is a quadratic residue of a prime $p \iff p$ is of the form $1 \pmod 8$ or $3 \pmod 8$ or $p = 2$.

In [18]:

```
# quick verification script!

for i in range(10000):
    # check that all anticipated values are quadratic residues
    if ((i % 8) == 1 or (i % 8) == 3 or i == 2) and is_prime(i):
        assert legendre(-2, i) == 1
        continue

    # make sure that all quadratic residues have been accounted for
    if is_prime(i) and legendre(-2, i) == 1:
        assert False
```