Euler’s Totient Theorem

Euler’s Totient function, also known as the **phi(n)** function, determines the count of the number of integers from 1 to n, which are coprime to n. Two integers are said to be coprime if their GCD is 1.

For Example: The values of phi(n) for some n are given below:

n = 1 => 1	n = 2 => 1	n = 3 => 2	n = 5 => 2	n = 5 => 4
n = 6 => 2	n = 7 => 6	n = 8 => 4	n = 9 => 6	n = 10 => 4

Properties of Euler Totient function:

- If p is prime, then gcd(p,q) = 1 for 1<=q<=p-1, thus phi(p) = p-1.
- If a and b are relatively prime then phi(a.b) = phi(a).phi(b)
- If p is prime, and k>=1, then there are exactly p^k/p numbers between 1 and p^k that are divisible by p, thus phi(p^k) = p^k - p^{k-1}.
- Generally, for non-coprime numbers x and y, phi(x.y) = phi(x).phi(y).(z/phi(z)) holds, where z is GCD(x,y).
- The sum of the values of the Euler totient function over all divisors of n equals n.

Finding the count of all the numbers from 1 to n that are coprime to n can be done by simply checking for each number from 1 to n, if the gcd of the number with n is 1, however, this leads to a linear time complexity in terms of n.
However, the same task can be efficiently performed by Euler’s totient function which is based upon Euler’s product formula which says that the value of phi(n) can be given as the product of the numbers obtained by multiplying n with (1 - 1/p), where p is a prime factor of n for each such prime factor p, in other words phi(n) = n*(1-1/p_1)*(1-1/p_2)*(1-1/p_3)…*(1-1/p_x), where p_1, p_2, p_3,…, p_x are prime factors of n.

Pseudocode:

```
/*
        Input n is a positive integer, returns the count of integers from 1 to n that are coprime to n
*/
function phi(n)

        //  Initialize the result to n
        res = n

        //  Finding the prime factorization of n
        for i = 2; i * i <=n; i++
                //  Checking if i is a prime factor of n
                if n mod i equals 0
                        //  Updating res
                        res = res - res / p
                        //  Updating n by eliminating all powers of prime i
                        while n mod i equals 0
                                n = n / i
        //   Check if n > 1, then n is also a prime divisor of the input
        if (n > 1)
                res = res - res / n

return res
```

Time complexity: O(sqrt(n)), where n is the given number.