# Euler's Totient Function - GeeksforGeeks

**Source:** https://www.geeksforgeeks.org/eulers-totient-function/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Euler's Totient Function Last Updated : 21 Jun, 2025 Given an integer n , find the value of Euler's Totient Function , denoted as Φ(n) . The function Φ(n) represents the count of positive integers less than or equal to n that are relatively prime to n . Euler's Totient function Φ(n) for an input n is the count of numbers in {1, 2, 3, ..., n-1} that are relatively prime to n, i.e., the numbers whose GCD (Greatest Common Divisor) with n is 1. If n is a positive integer and its prime factorization is; n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k} Where p_1, p_2, \ldots, p_k are distinct prime factors of n, then: \phi(n) = n \cdot \left(1 - \frac{1}{p_1}\right) \cdot \left(1 - \frac{1}{p_2}\right) \cdot \ldots \cdot \left(1 - \frac{1}{p_k}\right) . Try it on GfG Practice Examples: Input: n = 11 Output: 10 Explanation: From 1 to 11, 1,2,3,4,5,6,7,8,9,10 are relatively prime to 11. Input: n = 16 Output: 8 Explanation: From 1 to 16, 1,3,5,7,9,11,13,15 are relatively prime to 16. Table of Content [Naive Approach] Iterative GCD Method [Expected Approach] Euler’s Product Formula Some Interesting Properties of Euler's Totient Function [Naive Approach] Iterative GCD Method A simple solution is to iterate through all numbers from 1 to n-1 and count numbers with gcd with n as 1. Below is the implementation of the simple method to compute Euler's Totient function for an input integer n. C++ #include <iostream> using namespace std ; // Function to return gcd of a and b int gcd ( int a , int b ) { if ( a == 0 ) return b ; return gcd ( b % a , a ); } // A simple method to evaluate Euler Totient Function int etf ( int n ) { int result = 1 ; for ( int i = 2 ; i < n ; i ++ ) if ( gcd ( i , n ) == 1 ) result ++ ; return result ; } // Driver Code int main () { int n = 11 ; cout << etf ( n ); return 0 ; } Java class GfG { // Function to return gcd of a and b static int gcd ( int a , int b ) { if ( a == 0 ) return b ; return gcd ( b % a , a ); } // Function to compute Euler's Totient Function static int etf ( int n ) { int result = 1 ; for ( int i = 2 ; i < n ; i ++ ) { if ( gcd ( i , n ) == 1 ) result ++ ; } return result ; } // Driver Code public static void main ( String [] args ) { int n = 11 ; System . out . println ( etf ( n )); } } Python # Function to return gcd of a and b def gcd ( a , b ): if a == 0 : return b return gcd ( b % a , a ) # A simple method to evaluate Euler Totient Function def etf ( n ): result = 1 for i in range ( 2 , n ): if gcd ( i , n ) == 1 : result += 1 return result # Driver Code if __name__ == "__main__" : n = 11 print ( etf ( n )) C# using System ; class GfG { // Function to return gcd of a and b static int gcd ( int a , int b ) { if ( a == 0 ) return b ; return gcd ( b % a , a ); } // A simple method to evaluate Euler Totient Function static int etf ( int n ) { int result = 1 ; for ( int i = 2 ; i < n ; i ++ ){ if ( gcd ( i , n ) == 1 ) result ++ ; } return result ; } // Driver Code static void Main (){ int n = 11 ; Console . WriteLine ( etf ( n )); } } JavaScript // Function to return gcd of a and b function gcd ( a , b ) { if ( a === 0 ) return b ; return gcd ( b % a , a ); } // A simple method to evaluate Euler Totient Function function etf ( n ) { let result = 1 ; for ( let i = 2 ; i < n ; i ++ ) { if ( gcd ( i , n ) === 1 ) result ++ ; } return result ; } // Driver Code let n = 11 ; console . log ( etf ( n )); Output 10 Time Complexity: O(n log n) Auxiliary Space: O(log min(a,b)) where a,b are the parameters of gcd function. [Expected Approach] Euler’s Product Formula The idea is based on Euler's product formula which states that the value of totient functions is below the product overall prime factors p of n. 1) Initialize result as n 2) Consider every number 'p' (where 'p' varies from 2 to Φ(n)). If p divides n, then do following a) Subtract all multiples of p from 1 to n [all multiples of p will have gcd more than 1 (at least p) with n] b) Update n by repeatedly dividing it by p. 3) If the reduced n is more than 1, then remove all multiples of n from result. C++ #include <iostream> using namespace std ; int etf ( int n ){ int result = n ; // Consider all prime factors of n // and subtract their multiples // from result for ( int p = 2 ; p * p <= n ; ++ p ) { // Check if p is a prime factor. if ( n % p == 0 ) { // If yes, then update n and result while ( n % p == 0 ) n /= p ; result -= result / p ; } } // If n has a prime factor greater than sqrt(n) // (There can be at-most one such prime factor) if ( n > 1 ) result -= result / n ; return result ; } // Driver Code int main () { int n = 11 ; cout << etf ( n ); return 0 ; } Java class GfG { static int etf ( int n ) { int result = n ; // Consider all prime factors of n // and subtract their multiples // from result for ( int p = 2 ; p * p <= n ; ++ p ) { if ( n % p == 0 ) { while ( n % p == 0 ) n /= p ; result -= result / p ; } } // If n has a prime factor greater than sqrt(n) // (There can be at-most one such prime factor) if ( n > 1 ) result -= result / n ; return result ; } // Driver Code public static void main ( String [] args ) { int n = 11 ; System . out . println ( etf ( n )); } } Python def etf ( n ): result = n # Consider all prime factors of n # and subtract their multiples # from result p = 2 while p * p <= n : if n % p == 0 : while n % p == 0 : n //= p result -= result // p p += 1 # If n has a prime factor greater than sqrt(n) # (There can be at-most one such prime factor) if n > 1 : result -= result // n return result # Driver Code if __name__ == "__main__" : n = 11 print ( etf ( n )) C# using System ; class GfG { static int etf ( int n ) { int result = n ; // Consider all prime factors of n // and subtract their multiples // from result for ( int p = 2 ; p * p <= n ; ++ p ) { // Check if p is a prime factor. if ( n % p == 0 ) { // If yes, then update n and result while ( n % p == 0 ) n /= p ; result -= result / p ; } } // If n has a prime factor greater than sqrt(n) // (There can be at-most one such prime factor) if ( n > 1 ) result -= result / n ; return result ; } // Driver Code static void Main () { int n = 11 ; Console . WriteLine ( etf ( n )); } } JavaScript function etf ( n ) { let result = n ; // Consider all prime factors of n // and subtract their multiples // from result for ( let p = 2 ; p * p <= n ; ++ p ) { if ( n % p === 0 ) { while ( n % p === 0 ) n = Math . floor ( n / p ); result -= Math . floor ( result / p ); } } // If n has a prime factor greater than sqrt(n) // (There can be at-most one such prime factor) if ( n > 1 ) result -= Math . floor ( result / n ); return result ; } // Driver Code const n = 11 ; console . log ( etf ( n )); Output 10 Time Complexity: O(√n) Auxiliary Space: O(1) Some Interesting Properties of Euler's Totient Function 1) For a prime number p , \phi(p) = p - 1 Proof : \phi(p) = p - 1 , where p is any prime number We know that gcd(p, k) = 1 where k is any random number and k \neq p \\ Total number from 1 to p = p Number for which gcd(p, k) = 1 is 1 , i.e the number p itself, so subtracting 1 from p \phi(p) = p - 1 Examples : \phi(5) = 5 - 1 = 4 \\ \phi(13) = 13 - 1 = 12 \\ \phi(29) = 29 - 1 = 28 . 2) For two prime numbers a and b \phi(a \cdot b) = \phi(a) \cdot \phi(b) = (a - 1) \cdot (b - 1) , used in RSA Algorithm Proof : Let a and b be distinct primes. Then: φ(a)=a–1, φ(b)=b–1 Total numbers from 1 to ab: ab Multiples of a: b numbers Multiples of b: a numbers Common multiple (i.e., double-counted): only ab So, numbers not coprime to ab: a+b–1 Then, φ(ab) = ab – (a + b – 1) = ab – a – b + 1 = (a–1)(b–1) Hence, φ(ab) = φ(a)* φ(b) Examples : φ(5·7) = φ(5)·φ(7) = (5–1) (7–1) = 4·6 = 24 φ(3·5) = φ(3)·φ(5) = (3–1) (5–1) = 2·4 = 8 φ(3·7) = φ(3)·φ(7) = (3–1) (7–1) = 2·6 = 12 3) For a prime number p and integer k ≥ 1: φ(p k ) = p k –p k–1 Proof : \phi(p^k) = p ^ k - p ^{k - 1} , where p is a prime number \\ Total numbers from 1 to p ^ k = p ^ k Total multiples of p = \frac {p ^ k} {p} = p ^ {k - 1} Removing these multiples as with them gcd \neq 1 \\ Examples : p = 2, k = 5, p ^ k = 32 Multiples of 2 (as with them gcd \neq 1 ) = 32 / 2 = 16 \\ \phi(p ^ k) = p ^ k - p ^ {k - 1} . 4) Special Case : gcd(a, b) = 1 \phi(a \cdot b) = \phi(a) \cdot \phi(b) \cdot \frac {1} {\phi(1)} = \phi(a) \cdot \phi(b) . Examples : Special Case: gcd(a, b) = 1 , \phi(a \cdot b) = \phi(a) \cdot \phi(b) \phi(2 \cdot 9) = \phi(2) \cdot \phi(9) = 1 \cdot 6 = 6 \\ \phi(8 \cdot 9) = \phi(8) \cdot \phi(9) = 4 \cdot 6 = 24 \\ \phi(5 \cdot 6) = \phi(5) \cdot \phi(6) = 4 \cdot 2 = 8 Normal Case: gcd(a, b) \neq 1 , \phi(a \cdot b) = \phi(a) \cdot \phi(b) \cdot \frac {gcd(a, b)} {\phi(gcd(a, b))} \\ \phi(4 \cdot 6) = \phi(4) \cdot \phi(6) \cdot \frac {gcd(4, 6)} {\phi(gcd(4, 6))} = 2 \cdot 2 \cdot \frac{2}{1} = 2 \cdot 2 \cdot 2 = 8 \\ \phi(4 \cdot 8) = \phi(4) \cdot \phi(8) \cdot \frac {gcd(4, 8)} {\phi(gcd(4, 8))} = 2 \cdot 4 \cdot \frac{4}{2} = 2 \cdot 4 \cdot 2 = 16 \\ \phi(6 \cdot 8) = \phi(6) \cdot \phi(8) \cdot \frac {gcd(6, 8)} {\phi(gcd(6, 8))} = 2 \cdot 4 \cdot \frac{2}{1} = 2 \cdot 4 \cdot 2 = 16 . 5) Sum of values of totient functions of all divisors of n is equal to n. Example : n = 6 , factors = {1, 2, 3, 6} n = \phi(1) + \phi(2) + \phi(3) + \phi(6) = 1 + 1 + 2 + 2 = 6 6) The most famous and important feature is expressed in Euler's theorem : The theorem states that if n and a are coprime (or relatively prime) positive integers, then a Φ(n) Φ 1 (mod n) The RSA cryptosystem is based on this theorem: In the particular case when m is prime say p, Euler's theorem turns into the so-called Fermat's little theorem : a p-1 Φ 1 (mod p) Related Article: Euler’s Totient function for all numbers smaller than or equal to n Optimized Euler Totient Function for Multiple Evaluations Comment Article Tags: Article Tags: Mathematical DSA GCD-LCM sieve Modular Arithmetic Prime Number number-theory Numbers euler-totient + 5 More