Program to Find GCD or HCF of Two Numbers - GeeksforGeeks

Source: https://www.geeksforgeeks.org/program-to-find-gcd-or-hcf-of-two-numbers/

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 Program to Find GCD or HCF of Two Numbers Last Updated : 3 Oct, 2025 Given two positive integers a and b , the task is to find the GCD of the two numbers. Note: The GCD (Greatest Common Divisor) or HCF (Highest Common Factor) of two numbers is the largest number that divides both of them. Examples: Input: a = 20, b = 28 Output: 4 Explanation: The factors of 20 are 1, 2, 4, 5, 10 and 20. The factors of 28 are 1, 2, 4, 7, 14 and 28. Among these factors, 1, 2 and 4 are the common factors of both 20 and 28. The greatest among the common factors is 4. Input: a = 60, b = 36 Output: 12 Explanation: GCD of 60 and 36 is 12. Try it on GfG Practice Table of Content [Approach - 1] Using Loop - O(min(a, b)) Time and O(1) Space [Approach - 2] Euclidean Algorithm using Subtraction - O(min(a,b)) Time and O(min(a,b)) Space [Approach - 3] Modified Euclidean Algorithm using Subtraction by Checking Divisibility - O(min(a, b)) Time and O(min(a, b)) Space [Approach - 4] Optimized Euclidean Algorithm by Checking Remainder [Approach - 5] Using Built-in Function - O(log(min(a, b))) Time and O(1) Space [Approach - 1] Using Loop - O(min(a, b)) Time and O(1) Space The idea is to find the minimum of the two numbers and find its highest factor which is also a factor of the other number. C++ #include <iostream> using namespace std ; int gcd (int a , int b) { // Find Minimum of a and b int result = min (a , b); while (result > 0) { if (a % result == 0 && b % result == 0) { break ; } result -- ; } // Return gcd of a and b return result ; } int main () { int a = 20 , b = 28 ; cout << gcd (a , b); return 0 ; } Java import java.io.* ; class GFG { static int gcd (int a , int b) { // Find Minimum of a and b int result = Math . min (a , b); while (result > 0) { if (a % result == 0 && b % result == 0) { break ; } result -- ; } // Return gcd of a and b return result ; } public static void main (String [] args) { int a = 20 , b = 28 ; System . out . print (gcd (a , b)); } } Python def gcd (a , b): # Find Minimum of a and b result = min (a , b) while result > 0 : if a % result == 0 and b % result == 0 : break result -= 1 # Return gcd of a and b return result if __name__ == '__main__' : a = 20 b = 28 print (gcd (a , b)) C# using System ; class GFG { // Function to find gcd of two numbers static int gcd (int a , int b) { // Find Minimum of a and b int result = Math . Min (a , b); while (result > 0) { if (a % result == 0 && b % result == 0) break ; result -- ; } // Return gcd of a and b return result ; } static void Main () { int a = 20 ; int b = 28 ; Console . WriteLine (gcd (a , b)); } } JavaScript function gcd (a , b) { // Find Minimum of a and b let result = Math . min (a , b); while (result > 0) { if (a % result === 0 && b % result === 0) { break ; } result -- ; } // Return gcd of a and b return result ; } // Driver Code let a = 20 ; let b = 28 ; console . log (gcd (a , b)); Output 4 Below both approaches are optimized approaches of the above code. [Approach - 2] Euclidean Algorithm using Subtraction - O(min(a,b)) Time and O(min(a,b)) Space The idea of this algorithm is, the GCD of two numbers doesn't change if the smaller number is subtracted from the bigger number. This is the Euclidean algorithm by subtraction . It is a process of repeat subtraction, carrying the result forward each time until the result is equal to any one number being subtracted. Pseudo-code: gcd(a, b): if a = b: return a if a > b: return gcd(a - b, b) else: return gcd(a, b - a) C++ #include <iostream> using namespace std ; int gcd (int a , int b) { // Everything divides 0 if (a == 0) return b ; if (b == 0) return a ; // Base case if (a == b) return a ; // a is greater if (a > b) return gcd (a - b , b); return gcd (a , b - a); } int main () { int a = 20 , b = 28 ; cout << gcd (a , b); return 0 ; } Java class GfG { static int gcd (int a , int b) { // Everything divides 0 if (a == 0) return b ; if (b == 0) return a ; // Base case if (a == b) return a ; // a is greater if (a > b) return gcd (a - b , b); return gcd (a , b - a); } public static void main (String

[] args) { int a = 20 , b = 28 ; System . out . println (gcd (a , b)); } } Python def gcd (a , b): # Everything divides 0 if a == 0 : return b if b == 0 : return a # Base case if a == b : return a # a is greater if a > b : return gcd (a - b , b) return gcd (a , b - a) if __name__ == '__main__' : a = 20 b = 28 print (gcd (a , b)) C# using System ; class GfG { static int gcd (int a , int b) { // Everything divides 0 if (a == 0) return b ; if (b == 0) return a ; // Base case if (a == b) return a ; // a is greater if (a > b) return gcd (a - b , b); return gcd (a , b - a); } static void Main () { int a = 20 , b = 28 ; Console . WriteLine (gcd (a , b)); } } JavaScript function gcd (a , b) { // Everything divides 0 if (a === 0) return b ; if (b === 0) return a ; // Base case if (a === b) return a ; // a is greater if (a > b) return gcd (a - b , b); return gcd (a , b - a); } // Driver code let a = 20 , b = 28 ; console . log (gcd (a , b)); Output 4 [Approach - 3] Modified Euclidean Algorithm using Subtraction by Checking Divisibility - O(min(a, b)) Time and O(min(a, b)) Space The above method can be optimized based on the following idea: If we notice the previous approach, we can see at some point, one number becomes a factor of the other so instead of repeatedly subtracting till both become equal, we can check if it is a factor of the other. Illustration: See the below illustration for a better understanding: Consider a = 98 and b = 56 a = 98, b = 56: a > b so put a = a-b and b remains same. So a = 98-56 = 42 & b= 56. a = 42, b = 56: Since b > a, we check if b%a=0. Since answer is no, we proceed further. Now b>a. So b = b-a and a remains same. So b = 56-42 = 14 & a= 42. a = 42, b = 14: Since a>b, we check if a%b=0. Now the answer is yes. So we print smaller among a and b as H.C.F . i.e. 42 is 3 times of 14. So HCF is 14. C++ #include <iostream> using namespace std ; int gcd (int a , int b) { // Everything divides 0 if (a == 0) return b ; if (b == 0) return a ; // Base case if (a == b) return a ; // a is greater if (a > b) { if (a % b == 0) return b ; return gcd (a - b , b); } // b is greater if (b % a == 0) return a ; return gcd (a , b - a); } // Driver code int main () { int a = 20 , b = 28 ; cout << gcd (a , b); return 0 ; } Java class GfG { static int gcd (int a , int b) { // Everything divides 0 if (a == 0) return b ; if (b == 0) return a ; // Base case if (a == b) return a ; // a is greater if (a > b) { if (a % b == 0) return b ; return gcd (a - b , b); } // b is greater if (b % a == 0) return a ; return gcd (a , b - a); } // Driver code public static void main (String [] args) { int a = 20 , b = 28 ; System . out . println (gcd (a , b)); } } Python def gcd (a , b): # Everything divides 0 if a == 0 : return b if b == 0 : return a # Base case if a == b : return a # a is greater if a > b : if a % b == 0 : return b return gcd (a - b , b) # b is greater if b % a == 0 : return a return gcd (a , b - a) # Driver code if __name__ == '__main__' : a = 20 b = 28 print (gcd (a , b)) C# using System ; class GfG { static int gcd (int a , int b) { // Everything divides 0 if (a == 0) return b ; if (b == 0) return a ; // Base case if (a == b) return a ; // a is greater if (a > b) { if (a % b == 0) return b ; return gcd (a - b , b); } // b is greater if (b % a == 0) return a ; return gcd (a , b - a); } // Driver code static void Main () { int a = 20 , b = 28 ; Console . WriteLine (gcd (a , b)); } } JavaScript function gcd (a , b) { // Everything divides 0 if (a === 0) return b ; if (b === 0) return a ; // Base case if (a === b) return a ; // a is greater if (a > b) { if (a % b === 0) return b ; return gcd (a - b , b); } // b is greater if (b % a === 0) return a ; return gcd (a , b - a); } // Driver code let a = 20 , b = 28 ; console . log (gcd (a , b)); Output 4 [Approach - 4] Optimized Euclidean Algorithm by Checking Remainder Instead of the Euclidean algorithm by subtraction, a better approach can be used. We don't perform subtraction here. we continuously divide the bigger number by the smaller number. More can be learned about this efficient solution by using the modulo operator in Euclidean algorithm . C++ #include <iostream> using namespace std ; // Recursive function to calculate GCD using Euclidean algorithm int gcd (int a , int b) { return b == 0 ? a : gcd (b , a % b); } // Driver code int main () { int a = 20 , b = 28 ; cout << gcd (a , b); return 0 ; } Java class GfG { // Recursive function to calculate GCD using Euclidean algorithm static int gcd (int a , int b) { return (b == 0) ? a : gcd (b , a % b); } // Driver code public static void main (String [] args) { int a = 20 , b = 28 ; System . out . println (gcd (a , b)); } } Python # Recursive function to calculate GCD using Euclidean algorithm def gcd (a , b): return a if b == 0 else gcd (b , a % b) # Driver code a = 20 b = 28 print (gcd (a , b)) # Output: 4 C# using System ; class GfG { // Recursive function to calculate GCD using Euclidean algorithm static int gcd (int a , int b) => b == 0 ? a : gcd (b , a % b); // Driver code static void Main () { int a = 20 , b = 28 ; Console . WriteLine (gcd (a , b)); } } JavaScript // Recursive function to calculate GCD using Euclidean algorithm function gcd (a , b) { return b === 0 ? a : gcd (b , a % b); } // Driver code let a = 20 , b = 28 ; console . log (gcd (a , b)); Output 4 Time Complexity: O(log(min(a,b))) Each recursive call reduces the size of the numbers significantly using the modulo operation (a % b), which shrinks the input faster than subtraction. The worst-case scenario for the number of steps occurs when the inputs are consecutive Fibonacci numbers, like (21, 13), which maximizes the number of recursive calls. Since Fibonacci numbers grow exponentially, and the number of steps increases linearly with their position, the time complexity becomes logarithmic in terms of the smaller number — O(log(min(a, b))). Auxiliary Space: O(log(min(a,b)) The maximum number of recursive calls is proportional to the number of steps taken to reduce the input to zero, which is O(log(min(a, b))) in the worst case. [Approach - 5] Using Built-in Function - O(log(min(a, b))) Time and O(1) Space Languages like C++ have inbuilt functions to calculate GCD of two numbers. Below is the implementation using inbuilt functions. C++ #include <algorithm> #include <iostream> using namespace std ; int gcd (int a , int b) { return __gcd (a , b); } // Driver code int main () { int a = 20 , b = 28 ; cout << gcd (a , b); return 0 ; } Java import java.math.BigInteger ; class GfG { public static void main (String [] args) { int a = 20 , b = 28 ; // Convert to BigInteger and compute GCD int gcd = BigInteger . valueOf (a). gcd (BigInteger . valueOf (b)). intValue (); System . out . println (gcd); } } Python import math def gcd (a , b): return math . gcd (a , b) # Driver code if __name__ == '__main__' : a = 20 b = 28 print (gcd (a , b)) C# using System ; class GfG { static int GCD (int a , int b) { return b == 0 ? a : GCD (b , a % b); } // Driver code static void Main () { int a = 20 , b = 28 ; Console . WriteLine (GCD (a , b)); } } JavaScript function gcd (a , b) { return b === 0 ? a : gcd (b , a % b); } // Driver code let a = 20 , b = 28 ; console . log (gcd (a , b)); Output 4 Please refer GCD of more than two (or array) numbers to find HCF of more than two numbers. Comment Article Tags: Article Tags: Mathematical DSA Basic Coding Problems SAP Labs GCD-LCM + 1 More