Modular Exponentiation (Power in Modular Arithmetic) - GeeksforGeeks

Source: https://www.geeksforgeeks.org/modular-exponentiation-power-in-modular-arithmetic/

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 Modular Exponentiation (Power in Modular Arithmetic) Last Updated : 5 Feb, 2026 Given three integers x, n, and M, compute (x n) % M (remainder when x raised to the power n is divided by M). Examples : Input: x = 3, n = 2, M = 4 Output: 1 Explanation: 3 2 % 4 = 9 % 4 = 1. Input: x = 2, n = 6, M = 10 Output: 4 Explanation: 2 6 % 10 = 64 % 10 = 4. Try it on GfG Practice Table of Content [Naive Approach] Repeated Multiplication Method - O(n) Time and O(1) Space [Expected Approach] Modular Exponentiation Method - O(log(n)) Time and O(1) Space [Naive Approach] Repeated Multiplication Method - O(n) Time and O(1) Space We initialize the result as 1 and iterate from 1 to n, updating the result by multiplying it with x and taking the modulo by M in each step to keep the number within integer bounds. C++ #include <iostream> using namespace std ; int powMod (int x , int n , int M) { // Initialize result as 1 (since anything power 0 is 1) long res = 1 ; // n times to multiply x with itself for (int i = 1 ; i <= n ; i ++) { // Multiplying res with x // and taking modulo to avoid overflow res = (res * x) % M ; } return res ; } int main () { int x = 3 , n = 2 , M = 4 ; cout << powMod (x , n , M) << endl ; return 0 ; } Java public class GfG { public static int powMod (int x , int n , int M) { Long res = 1 ; for (int i = 1 ; i <= n ; i ++) { // Multiplying res with x and // taking modulo to avoid overflow res = (res * x) % M ; } return res ; } public static void main (String [] args) { int x = 3 , n = 2 , M = 4 ; System . out . println (powMod (x , n , M)); } } Python def powMod (x , n , M): res = 1 # loop from 1 to n for _ in range (n): # Multiplying res with x # and taking modulo to avoid overflow res = (res * x) % M return res if __name__ == "__main__" : x , n , M = 3 , 2 , 4 print (powMod (x , n , M)) C# using System ; public class GfG { public static int PowMod (int x , int n , int M) { long res = 1 ; for (int i = 1 ; i <= n ; i ++) { // Multiplying res with x and taking modulo to avoid overflow res = (res * x) % M ; } return (int) res ; } public static void Main (string [] args) { int x = 3 , n = 2 , M = 4 ; Console . WriteLine (PowMod (x , n , M)); } } JavaScript function powMod (x , n , M) { let res = 1 ; // Loop n times, multiplying x and taking modulo at each step for (let i = 1 ; i <= n ; i ++) { // Multiplying res with x and // taking modulo to avoid overflow res = (res * x) % M ; } return res ; } // Driver Code let x = 3 , n = 2 , M = 4 ; console . log (powMod (x , n , M)); Output 1 [Expected Approach] Modular Exponentiation Method - O(log(n)) Time and O(1) Space The idea of binary exponentiation is to reduce the exponent by half at each step, using squaring, which lowers the time complexity from O(n) to O(log n). -> x n = (x n/2) 2 if n is even. -> x n = x*x n-1 if n is odd. Step by step approach: Start with the result as 1. Use a loop that runs while the exponent n is greater than 0. If the current exponent is odd, multiply the result by the current base and apply the modulo. Square the base and take the modulo to keep the value within bounds. Divide the exponent by 2 (ignore the remainder). Repeat the process until the exponent becomes 0. C++ #include <iostream> using namespace std ; int powMod (int x , int n , int M) { int res = 1 ; // Loop until exponent becomes 0 while (n >= 1) { // n is odd, multiply result by current x and take modulo if (n & 1) { res = (res * x) % M ; // Reduce exponent by 1 to make it even n -- ; } // n is even, square the base and halve the exponent else { x = (x * x) % M ; n /= 2 ; } } return res ; } int main () { int x = 3 , n = 2 , M = 4 ; cout << powMod (x , n , M) << endl ; } Java class GfG { public int powMod (int x , int n , int M) { int res = 1 ; // Loop until exponent becomes 0 while (n >= 1) { // n is odd, multiply result by current x and take modulo if ((n & 1) == 1) { res = (res * x) % M ; // Decrease n to make it even n -- ; } else { // n is even, square the base and halve the exponent x = (x * x) % M ; n /= 2 ; } } return res ; } public static void main (String [] args)

{ int x = 3 , n = 2 , M = 4 ; GfG obj = new GfG (); System . out . println (obj . powMod (x , n , M)); } } Python def powMod (x , n , M): res = 1 # Loop until exponent becomes 0 while n >= 1 : # n is odd, multiply result by current x and take modulo if n % 2 == 1 : res = (res * x) % M # Make n even n -= 1 else : # n is even, square the base and halve the exponent x = (x * x) % M n //= 2 return res if __name__ == "__main__" : x , n , M = 3 , 2 , 4 print (powMod (x , n , M)) C# using System ; class GfG { public int powMod (int x , int n , int M) { int res = 1 ; // Loop until exponent becomes 0 while (n >= 1) { // n is odd, multiply result by current x and take modulo if ((n & 1) == 1) { res = (int)((1L * res * x) % M); // Reduce exponent by 1 n -- ; } else { // n is even, square the base and halve the exponent x = (int)((1L * x * x) % M); n /= 2 ; } } return res ; } public static void Main () { int x = 3 , n = 2 , M = 4 ; GfG obj = new GfG (); Console . WriteLine (obj . powMod (x , n , M)); } } JavaScript function powMod (x , n , M) { let res = 1 ; // Loop until exponent becomes 0 while (n >= 1) { // If n is odd, multiply result by current x and take modulo if (n % 2 === 1) { res = (res * x) % M ; n -= 1 ; } else { // If n is even, square the base and halve the exponent x = (x * x) % M ; n /= 2 ; } } return res ; } // Driver Code let x = 3 , n = 2 , M = 4 ; console . log (powMod (x , n , M)); Output 1 Comment Article Tags: Article Tags: Divide and Conquer Mathematical DSA Google Modular Arithmetic large-numbers + 2 More