Count set bits in an integer - GeeksforGeeks

Source: https://www.geeksforgeeks.org/count-set-bits-in-an-integer/

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 Count set bits in an integer Last Updated : 1 Oct, 2025 Given a positive number n count the set bit . Examples : Input : n = 6 Output : 2 Binary representation of 6 is 110 and has 2 set bits Input : n = 13 Output : 3 Binary representation of 13 is 1101 and has 3 set bits Try it on GfG Practice Table of Content [Naive Approach] - One by One Counting [Expected Approach 1] - Brian Kernighan's Algorithm [Expected Approach 2] - Using Lookup table [Expected Approach 3] - Mapping Nibbles Using power of 2 (efficient method to find for large value also) Using Library or Quick Syntax [Naive Approach] - One by One Counting This approach counts the set bits by checking each bit from right to left. It uses n & 1 to check if the least significant bit is 1, increments the count if so, then right-shifts the number to check the next bit. This repeats until n becomes 0, returning the total number of set bits. C++ #include <bits/stdc++.h> using namespace std ; /* Function to get no of set bits in binary representation of positive integer n */ unsigned int countSetBits (unsigned int n) { unsigned int count = 0 ; while (n) { count += n & 1 ; n >>= 1 ; } return count ; } int main () { int i = 9 ; cout << countSetBits (i); return 0 ; } C #include <stdio.h> /* Function to get no of set bits in binary representation of positive integer n */ unsigned int countSetBits (unsigned int n) { unsigned int count = 0 ; while (n) { count += n & 1 ; n >>= 1 ; } return count ; } int main () { int i = 9 ; printf ("%d" , countSetBits (i)); return 0 ; } Java import java.io.* ; class GFG { /* Function to get no of set bits in binary representation of positive integer n */ static int countSetBits (int n) { int count = 0 ; while (n > 0) { count += n & 1 ; n >>= 1 ; } return count ; } public static void main (String args []) { int i = 9 ; System . out . println (countSetBits (i)); } } Python # Function to get no of set # bits in binary representation # of positive integer n def countSetBits (n): count = 0 while (n): count += n & 1 n >>= 1 return count # Program to test function countSetBits if __name__ == "__main__" : i = 9 print (countSetBits (i)) C# using System ; class GFG { // Function to get no of set // bits in binary representation // of positive integer n static int countSetBits (int n) { int count = 0 ; while (n > 0) { count += n & 1 ; n >>= 1 ; } return count ; } public static void Main () { int i = 9 ; Console . Write (countSetBits (i)); } } JavaScript // Javascript program to Count set // bits in an integer /* Function to get no of set bits in binary representation of positive integer n */ function countSetBits (n) { var count = 0 ; while (n) { count += n & 1 ; n >>= 1 ; } return count ; } // Driver Code var i = 9 ; console . log (countSetBits (i)); PHP <?php // Function to get no of set // bits in binary representation // of positive integer n function countSetBits ($n) { $count = 0 ; while ($n) { $count += $n & 1 ; $n >>= 1 ; } return $count ; } $i = 9 ; echo countSetBits ($i); ?> Output 2 Time Complexity: O(log n) Auxiliary Space: O(1) Recursive Implementation The function checks if the least significant bit is 1 and adds to the count. It then right-shifts the number and repeats this process recursively until all bits are checked, returning the total count of set bits. C++ #include <bits/stdc++.h> using namespace std ; // recursive function to count set bits int countSetBits (int n) { // base case if (n == 0) return 0 ; else // if last bit set add 1 else add 0 return (n & 1) + countSetBits (n >> 1); } // driver code int main () { int n = 9 ; // function calling cout << countSetBits (n); return 0 ; } C #include <stdio.h> // recursive function to count set bits int countSetBits (int n) { // base case if (n == 0) return 0 ; else // if last bit set add 1 else add 0 return (n & 1) + countSetBits (n >> 1); } // driver code int main () { int n = 9 ; // function calling printf ("%d" , countSetBits (n)); return 0 ; } Java import java.io.* ; class GFG { // recursive function to count set bits public static int countSetBits (int n) { // base case if (n == 0) return 0 ; else // if last bit set add 1 else add 0 return (n & 1) + countSetBits (n >> 1); } // Driver code public static void main (String [] args) { // get value from user int n = 9 ; // function calling System . out . println (countSetBits (n)); } } Python

def countSetBits (n): # base case if (n == 0): return 0 else : # if last bit set add 1 else # add 0 return (n & 1) + countSetBits (n >> 1) # Get value from user n = 9 # Function calling print (countSetBits (n)) C# using System ; class GFG { // recursive function // to count set bits public static int countSetBits (int n) { // base case if (n == 0) return 0 ; else // if last bit set // add 1 else add 0 return (n & 1) + countSetBits (n >> 1); } // Driver code static public void Main () { // get value // from user int n = 9 ; // function calling Console . WriteLine (countSetBits (n)); } } JavaScript // recursive function to count set bits function countSetBits (n) { // base case if (n == 0) return 0 ; else // if last bit set add 1 else add 0 return (n & 1) + countSetBits (n >> 1); } // driver code let n = 9 console . log (countSetBits (n)); PHP <?php // recursive function // to count set bits function countSetBits ($n) { // base case if ($n == 0) return 0 ; else // if last bit set // add 1 else add 0 return ($n & 1) + countSetBits ($n >> 1); } $n = 9 ; // function calling echo countSetBits ($n); ?> Output 2 Time Complexity: O(log n) Auxiliary Space: O(log n)for recursive stack space [Expected Approach 1] - Brian Kernighan's Algorithm Brian Kernighan’s algorithm counts set bits efficiently by clearing them one by one. The key operation is n &= (n - 1), which removes the least significant set bit from n. Each time this is done, it means one set bit has been counted and removed. The process repeats until n becomes 0, and the total number of times the loop runs gives the count of set bits. This is faster than checking each bit individually because it only iterates as many times as there are set bits in the number. Subtracting 1 from a decimal number flips all the bits after the rightmost set bit(which is 1) including the rightmost set bit. for example : 10 in binary is 00001010 9 in binary is 00001001 8 in binary is 00001000 7 in binary is 00000111 So if we subtract a number by 1 and do it bitwise & with itself (n & (n-1)), we unset the rightmost set bit. If we do n & (n-1) in a loop and count the number of times the loop executes, we get the set bit count. The beauty of this solution is the number of times it loops is equal to the number of set bits in a given integer. 1 Initialize count: = 0 2 If integer n is not zero (a) Do bitwise & with (n-1) and assign the value back to n n: = n&(n-1) (b) Increment count by 1 (c) go to step 2 3 Else return count C++ #include <iostream> using namespace std ; class gfg { /* Function to get no of set bits in binary representation of passed binary no. */ public : unsigned int countSetBits (int n) { unsigned int count = 0 ; while (n) { n &= (n - 1); count ++ ; } return count ; } }; /* Program to test function countSetBits */ int main () { gfg g ; int i = 9 ; cout << g . countSetBits (i); return 0 ; } C #include <stdio.h> /* Function to get no of set bits in binary representation of passed binary no. */ unsigned int countSetBits (int n) { unsigned int count = 0 ; while (n) { n &= (n - 1); count ++ ; } return count ; } /* Program to test function countSetBits */ int main () { int i = 9 ; printf ("%d" , countSetBits (i)); getchar (); return 0 ; } Java import java.io.* ; class GFG { /* Function to get no of set bits in binary representation of passed binary no. */ static int countSetBits (int n) { int count = 0 ; while (n > 0) { n &= (n - 1); count ++ ; } return count ; } public static void main (String args []) { int i = 9 ; System . out . println (countSetBits (i)); } } // This code is contributed by Anshika Goyal. Python # Function to get no of set bits in binary # representation of passed binary no. */ def countSetBits (n): count = 0 while (n): n &= (n - 1) count += 1 return count # Program to test function countSetBits i = 9 print (countSetBits (i)) C# using System ; class GFG { /* Function to get no of set bits in binary representation of passed binary no. */ static int countSetBits (int n) { int count = 0 ; while (n > 0) { n &= (n - 1); count ++ ; } return count ; } static public void Main () { int i = 9 ; Console . WriteLine (countSetBits (i)); } } // This code is contributed by ajit JavaScript /* Function to get no of set bits in binary representation of passed binary no. */ function countSetBits (n) { var count = 0 ; while (n > 0) { n &= (n - 1); count ++ ; } return count ; } // driver program var i = 9 ; console . log (countSetBits (i)); PHP <?php /* Function to get no of set bits in binary representation of passed binary no. */ function countSetBits ($n) { $count = 0 ; while ($n) { $n &= ($n - 1) ; $count ++ ; } return $count ; } $i = 9 ; echo countSetBits ($i); ?> Output 2 Time Complexity: O(log n) Auxiliary Space: O(1) [Expected Approach 2] - Using Lookup table It first builds a table BitsSetTable256[] storing the count of set bits for all 8-bit numbers (0–255). In countSetBits, the 32-bit number is divided into four 8-bit chunks, and the table is used to quickly find the set bits in each byte. Summing these gives the total set bit count. This method is efficient because it replaces bitwise loops with quick table lookups, making it ideal when frequent set bit counting is needed. C++ #include <bits/stdc++.h> using namespace std ; int BitsSetTable256 [256]; // Function to initialise the lookup table void initialize () { // To initially generate the // table algorithmically BitsSetTable256 [0] = 0 ; for (int i = 0 ; i < 256 ; i ++) { BitsSetTable256 [i] = (i & 1) + BitsSetTable256 [i / 2]; } } // Function to return the count // of set bits in n int countSetBits (int n) { return (BitsSetTable256 [n & 0xff] + BitsSetTable256 [(n >> 8) & 0xff] + BitsSetTable256 [(n >> 16) & 0xff] + BitsSetTable256 [n >> 24]); } int main () { // Initialise the lookup table initialize (); int n = 9 ; cout << countSetBits (n); } Java import java.util.* ; class GFG { static int [] BitsSetTable256 = new int [256] ; // Function to initialise the lookup table public static void initialize () { // To initially generate

the // table algorithmically BitsSetTable256 [0] = 0 ; for (int i = 0 ; i < 256 ; i ++) { BitsSetTable256 [i] = (i & 1) + BitsSetTable256 [i / 2] ; } } // Function to return the count // of set bits in n public static int countSetBits (int n) { return (BitsSetTable256 [n & 0xff] + BitsSetTable256 [(n >> 8) & 0xff] + BitsSetTable256 [(n >> 16) & 0xff] + BitsSetTable256 [n >> 24]); } // Driver code public static void main (String [] args) { // Initialise the lookup table initialize (); int n = 9 ; System . out . print (countSetBits (n)); } } Python BitsSetTable256 = [0] * 256 # Function to initialise the lookup table def initialize (): # To initially generate the # table algorithmically BitsSetTable256 [0] = 0 for i in range (256): BitsSetTable256 [i] = (i & 1) + BitsSetTable256 [i // 2] # Function to return the count # of set bits in n def countSetBits (n): return (BitsSetTable256 [n & 0xff] + BitsSetTable256 [(n >> 8) & 0xff] + BitsSetTable256 [(n >> 16) & 0xff] + BitsSetTable256 [n >> 24]) if __name__ == "__main__" : initialize () n = 9 print (countSetBits (n)) C# using System ; using System.Collections.Generic ; class GFG { static int [] BitsSetTable256 = new int [256]; // Function to initialise the lookup table public static void initialize () { // To initially generate the // table algorithmically BitsSetTable256 [0] = 0 ; for (int i = 0 ; i < 256 ; i ++) { BitsSetTable256 [i] = (i & 1) + BitsSetTable256 [i / 2]; } } // Function to return the count // of set bits in n public static int countSetBits (int n) { return (BitsSetTable256 [n & 0 xff] + BitsSetTable256 [(n >> 8) & 0 xff] + BitsSetTable256 [(n >> 16) & 0 xff] + BitsSetTable256 [n >> 24]); } public static void Main (String [] args) { // Initialise the lookup table initialize (); int n = 9 ; Console . Write (countSetBits (n)); } } JavaScript var BitsSetTable256 = Array . from ({ length : 256 }, (_ , i) => 0); // Function to initialise the lookup table function initialize () { // To initially generate the // table algorithmically BitsSetTable256 [0] = 0 ; for (var i = 0 ; i < 256 ; i ++) { BitsSetTable256 [i] = (i & 1) + BitsSetTable256 [parseInt (i / 2)]; } } // Function to return the count // of set bits in n function countSetBits (n) { return (BitsSetTable256 [n & 0xff] + BitsSetTable256 [(n >> 8) & 0xff] + BitsSetTable256 [(n >> 16) & 0xff] + BitsSetTable256 [n >> 24]); } // Driver code initialize (); var n = 9 ; console . log (countSetBits (n)); Output 2 Time Complexity: O(1) Auxiliary Space: O(1) [Expected Approach 3] - Mapping Nibbles It simply maintains a map(or array) of numbers to bits for a nibble. A Nibble contains 4 bits. So we need an array of up to 15. int num_to_bits[16] = {0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4}; Now we just need to get nibbles of a given long/int/word etc recursively. C++ #include <bits/stdc++.h> using namespace std ; int num_to_bits [16] = { 0 , 1 , 1 , 2 , 1 , 2 , 2 , 3 , 1 , 2 , 2 , 3 , 2 , 3 , 3 , 4 }; /* Recursively get nibble of a given number and map them in the array */ unsigned int countSetBitsRec (unsigned int num) { int nibble = 0 ; if (0 == num) return num_to_bits [0]; // Find last nibble nibble = num & 0xf ; // Use pre-stored values to find count // in last nibble plus recursively add // remaining nibbles. return num_to_bits [nibble] + countSetBitsRec (num >> 4); } int main () { int num = 9 ; cout << countSetBitsRec (num); return 0 ; } C #include <stdio.h> int num_to_bits [16] = { 0 , 1 , 1 , 2 , 1 , 2 , 2 , 3 , 1 , 2 , 2 , 3 , 2 , 3 , 3 , 4 }; /* Recursively get nibble of a given number and map them in the array */ unsigned int countSetBitsRec (unsigned int num) { int nibble = 0 ; if (0 == num) return num_to_bits [0]; // Find last nibble nibble = num & 0xf ; // Use pre-stored values to find count // in last nibble plus recursively add // remaining nibbles. return num_to_bits [nibble] + countSetBitsRec (num >> 4); } int main () { int num = 9 ; printf ("%d \n " , countSetBitsRec (num)); } Java import java.util.* ; class GFG { static int [] num_to_bits = new int [] { 0 , 1 , 1 , 2 , 1 , 2 , 2 , 3 , 1 , 2 , 2 , 3 , 2 , 3 , 3 , 4 }; static int countSetBitsRec (int num) { int nibble = 0 ; if (0 == num) return num_to_bits [0] ; // Find last nibble nibble = num & 0xf ; // Use pre-stored values to find count // in last nibble plus recursively add // remaining nibbles. return num_to_bits [nibble] + countSetBitsRec (num >> 4); } public static void main (String [] args) { int num = 9 ; System . out . println (countSetBitsRec (num)); } } Python num_to_bits = [0 , 1 , 1 , 2 , 1 , 2 , 2 , 3 , 1 , 2 , 2 , 3 , 2 , 3 , 3 , 4]; # Recursively get nibble of a given number # and map them in the array def countSetBitsRec (num): nibble = 0 ; if (0 == num): return num_to_bits [0]; # Find last nibble nibble = num & 0xf ; # Use pre-stored values to find count # in last nibble plus recursively add # remaining nibbles. return num_to_bits [nibble] + countSetBitsRec (num >> 4); # Driver code num = 9 ; print (countSetBitsRec (num)); # this code is contributed by mits C# class GFG { static int [] num_to_bits = new int [16] { 0 , 1 , 1 , 2 , 1 , 2 , 2 , 3 , 1 , 2 , 2 , 3 , 2 , 3 , 3 , 4 }; /* Recursively get nibble of a given number and map them in the array */ static int countSetBitsRec (int num) { int nibble = 0 ; if (0 == num) return num_to_bits [0]; // Find last nibble nibble = num & 0 xf ; // Use pre-stored values to find count // in last nibble plus recursively add // remaining nibbles. return num_to_bits [nibble] + countSetBitsRec (num >> 4); } // Driver code static void Main () { int num = 9 ; System . Console . WriteLine (countSetBitsRec (num)); } } // this code is contributed by mits JavaScript var num_to_bits = [0 , 1 , 1 , 2 , 1 , 2 , 2 , 3 , 1 , 2 , 2 , 3 , 2 , 3 , 3 , 4]; /* Recursively get nibble of a given number and map them in the array */ function countSetBitsRec (num) { var nibble = 0 ; if (0 == num) return num_to_bits [0]; // Find last nibble nibble = num & 0xf ; // Use pre-stored values

to find count // in last nibble plus recursively add // remaining nibbles. return num_to_bits [nibble] + countSetBitsRec (num >> 4); } // Driver code var num = 9 ; console . log (countSetBitsRec (num)); PHP <?php $num_to_bits = array (0 , 1 , 1 , 2 , 1 , 2 , 2 , 3 , 1 , 2 , 2 , 3 , 2 , 3 , 3 , 4); /* Recursively get nibble of a given number and map them in the array */ function countSetBitsRec ($num) { global $num_to_bits ; $nibble = 0 ; if (0 == $num) return $num_to_bits [0]; // Find last nibble $nibble = $num & 0xf ; // Use pre-stored values to find count // in last nibble plus recursively add // remaining nibbles. return $num_to_bits [$nibble] + countSetBitsRec ($num >> 4); } // Driver code $num = 9 ; echo (countSetBitsRec ($num)); // This code is contributed by mits ?> Output 2 Time Complexity: O(log n), because we have log(16, n) levels of recursion. Storage Complexity: O(1) Whether the given number is short, int, long, or long long we require an array of 16 sizes only, which is constant. Using power of 2 (efficient method to find for large value also) Iterate from k to 0 , where k is the largest power of 2 such that pow(2, k) <= num . And check if the Bitwise AND of num and pow(2, i) is greater than zero or not. If it is greater than zero , Then i-th bit is set ,then increase the count by 1. C++ #include <bits/stdc++.h> using namespace std ; // Function to find largest power of 2 such that // pow(2,k) <= N int findk (int n) { int k ; int i = 0 ; int val = pow (2 , i); while (val <= n) { k = i ; i ++ ; val = pow (2 , i); } return k ; } // Function to count set bits in a number int countSetBits (int N) { int count = 0 ; int k = findk (N); int val , x ; // Iterating from largest power to 2 such that // pow(2,k) to 0 for (int i = k ; i >= 0 ; i --) { val = pow (2 , i); x = val & N ; //x will store Bitwise AND of N & val if (x > 0) { count ++ ; } } return count ; //return count of set bits } int main () { int N = 9 ; // Function call cout << countSetBits (N) << endl ; return 0 ; } Java import java.io.* ; public class GFG { // Function to find largest power of 2 such that // pow(2,k) <= N static int findk (int n) { int k = 0 ; int i = 0 ; int val = (int) Math . pow (2 , i); while (val <= n) { k = i ; i ++ ; val = (int) Math . pow (2 , i); } return k ; } // Function to count set bits in a number static int countSetBits (int N) { int count = 0 ; int k = findk (N); int val , x ; // Iterating from largest // power to 2 such that // pow(2,k) to 0 for (int i = k ; i >= 0 ; i --) { val = (int) Math . pow (2 , i); x = val & N ; // x will store Bitwise AND of N & val if (x > 0) { count ++ ; } } return count ; // return count of set bits } public static void main (String [] args) { int N = 9 ; System . out . println (countSetBits (N)); } } Python import math # Function to find largest # power of 2 such that # pow(2,k) <= N def findk (n): i = 0 val = math . pow (2 , i) while val <= n : k = i i += 1 val = math . pow (2 , i) return k # Function to count set bits in a number def countSetBits (N): count = 0 k = findk (N) for i in range (k , - 1 , - 1): val = int (math . pow (2 , i)) x = val & N # x will store Bitwise AND of N & val if x > 0 : count += 1 return count if __name__ == '__main__' : N = 9 print (countSetBits (N)) C# using System ; class Program { // Function to find largest power of 2 such that // pow(2,k) <= N static int FindK (int n) { int k = 0 ; int i = 0 ; int val = (int) Math . Pow (2 , i); while (val <= n) { k = i ; i ++ ; val = (int) Math . Pow (2 , i); } return k ; } // Function to count set bits in a numnber static int CountSetBits (int N) { int count = 0 ; int k = FindK (N); int val , x ; // Iterating from largest power to 2 such that // pow(2,k) to 0 for (int i = k ; i >= 0 ; i --) { val = (int) Math . Pow (2 , i); x = val & N ; //x will store Bitwise AND of N & val if (x > 0) { count ++ ; } } return count ; //return count of set bits } static void Main () { int N = 9 ; // Function call Console . WriteLine (CountSetBits (N)); } } JavaScript // function to find largest power of 2 such that // pow(2,k) <= N function findk (n){ let k ; let i = 0 ; let val = Math . pow (2 , i); while (val <= n){ k = i ; i ++ ; val = Math . pow (2 , i); } return k ; } // function to count set bits in a number function countSetBits (N){ let count = 0 ; let k = findk (N); let val ; let x ; // iterating from largest power to 2 such that // pow(2,k) to 0 for (let i = k ; i >= 0 ; i --){ val = Math . pow (2 , i); x = val & N ; // x will store bitwise and of N and val if (x > 0) count ++ ; } // return count of set bits return count ; } // driver program let N = 9 ; console . log (countSetBits (N)); Output 2 Time Complexity: O(logn) Auxiliary Space: O(1) Using Library or Quick Syntax In C++ GCC, we can directly count set bits using __builtin_popcount(). So we can avoid a separate function for counting set bits. Similarly, we have direct syntax in other programming languages. C++ // C++ program to demonstrate __builtin_popcount() #include <iostream> using namespace std ; int main () { cout << __builtin_popcount (9); return 0 ; } Java // java program to demonstrate // __builtin_popcount() import java.io.* ; class GFG { // Driver code public static void main (String [] args) { System . out . println (Integer . bitCount (9)); } } Python print (bin (9) . count ('1')); C# using System ; class Program { static void Main (string [] args) { Console . WriteLine (Convert . ToString (9 , 2). Replace ("0" , ""). Length); } } JavaScript // Javascript program to demonstrate // __builtin_popcount() console . log ((9). toString (2). split (''). filter (x => x == '1'). length + "
"); PHP <?php // PHP program to demonstrate popcount equivalent echo substr_count (decbin (9), '1'); ?> Output 4 Comment Article Tags: Article Tags: Bit Magic DSA setBitCount Adobe Qualcomm Samsung Cisco Brocade Juniper Networks Wipro + 6 More