# Count total set bits in all numbers from 1 to n

Given a positive integer n, count the total number of set bits in binary representation of all numbers from 1 to n.

Examples:

```
Input: n = 3
Output:  4

Input: n = 6
Output: 9

Input: n = 7
Output: 12

Input: n = 8
Output: 13
```

Source: Amazon Interview Question

**Method 1 (Simple)**
A simple solution is to run a loop from 1 to n and sum the count of set bits in all numbers from 1 to n.

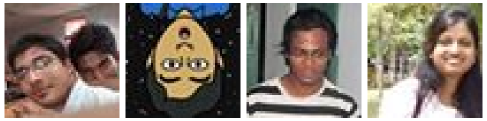
```
// A simple program to count set bits in all numbers from 1 to n.
#include <stdio.h>

// A utility function to count set bits in a number x
unsigned int countSetBitsUtil(unsigned int x);
```

Google™ Custom Search

GeeksforGeeks

53,526 people like GeeksforGeeks.



Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion
Geometric Algorithms

Geometric Algorithms

```
// Returns count of set bits present in all numbers from 1 to n
unsigned int countSetBits(unsigned int n)
{
    int bitCount = 0; // initialize the result

    for(int i = 1; i <= n; i++)
       bitCount += countSetBitsUtil(i);

    return bitCount;
}

// A utility function to count set bits in a number x
unsigned int countSetBitsUtil(unsigned int x)
{
    if (x <= 0)
        return 0;
    return (x %2 == 0? 0: 1) + countSetBitsUtil (x/2);
}

// Driver program to test above functions
int main()
{
   int n = 4;
   printf ("Total set bit count is %d", countSetBits(n));
   return 0;
}
```

Output:

```
Total set bit count is 6
```

Time Complexity: O(nLogn)

**Method 2 (Tricky)**
If the input number is of the form 2^b -1 e.g., 1,3,7,15.. etc, the number of set bits is b * 2^(b-1). This is because for all the numbers 0 to (2^b)-1, if you complement and flip the list you end up with the same list (half the bits are on, half off).

If the number does not have all set bits, then some position m is the position of leftmost set bit. The number of set bits in that position is n – (1 << m) + 1. The remaining set bits are in two parts:

Popular Posts

All permutations of a given string
Memory Layout of C Programs
Understanding “extern” keyword in C
Median of two sorted arrays
Tree traversal without recursion and without stack!
Structure Member Alignment, Padding and Data Packing
Intersection point of two Linked Lists
Lowest Common Ancestor in a BST.
Check if a binary tree is BST or not
Sorted Linked List to Balanced BST

1) The bits in the (m-1) positions down to the point where the leftmost bit becomes 0, and
2) The 2^(m-1) numbers below that point, which is the closed form above.

An easy way to look at it is to consider the number 6:

```
0|0 0
0|0 1
0|1 0
0|1 1
-|-
1|0 0
1|0 1
1|1 0
```

The leftmost set bit is in position 2 (positions are considered starting from 0). If we mask that off what remains is 2 (the "1 0" in the right part of the last row.) So the number of bits in the 2nd position (the lower left box) is 3 (that is, 2 + 1). The set bits from 0-3 (the upper right box above) is 2*2^(2-1) = 4. The box in the lower right is the remaining bits we haven't yet counted, and is the number of set bits for all the numbers up to 2 (the value of the last entry in the lower right box) which can be figured recursively.

```
// A O(Logn) complexity program to count set bits in all numbers from
#include <stdio.h>

/* Returns position of leftmost set bit. The rightmost
   position is considered as 0 */
unsigned int getLeftmostBit (int n)
{
   int m = 0;
   while (n  > 1)
   {
      n = n >> 1;
      m++;
   }
   return m;
}

/* Given the position of previous leftmost set bit in n (or an upper
   bound on leftmost position) returns the new position of leftmost
   set bit in n  */
unsigned int getNextLeftmostBit (int n, int m)
{
   unsigned int temp = 1 << m;
   while (n  < temp)
   {
      temp = temp >> 1;
      m--;
   }
   return m;
}

// The main recursive function used by countSetBits()
unsigned int _countSetBits(unsigned int n, int m);

// Returns count of set bits present in all numbers from 1 to n
unsigned int countSetBits(unsigned int n)
{
   // Get the position of leftmost set bit in n. This will be
   // used as an upper bound for next set bit function
   int m = getLeftmostBit (n);

   // Use the position
   return _countSetBits (n, m);
}

unsigned int _countSetBits(unsigned int n, int m)
{
    // Base Case: if n is 0, then set bit count is 0
    if (n == 0)
       return 0;

    /* get position of next leftmost set bit */
    m = getNextLeftmostBit(n, m);

    // If n is of the form 2^x-1, i.e., if n is like 1, 3, 7, 15, 31,.
    // then we are done.
    // Since positions are considered starting from 0, 1 is added to m
    if (n == ((unsigned int)1<<(m+1))-1)
        return (unsigned int)(m+1)*(1<<m);

    // update n for next recursive call
    n = n - (1<<m);
    return (n+1) + countSetBits(n) + m*(1<<(m-1));
}

// Driver program to test above functions
int main()
{
   int n = 17;
```

## Recent Comments

Abhi You live US or India?

Google (Mountain View) interview · 23 minutes ago

**Aman** Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 1 hour ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 1 hour ago

**Sanjay Agarwal** bool tree::Root_to_leaf_path_given_sum(tree...

Root to leaf path sum equal to a given number · 1 hour ago

**GOPI GOPINATH** @admin Highlight this sentence "We can easily...

Count trailing zeroes in factorial of a number · 1 hour ago

**newCoder3006** If the array contains negative numbers also. We...

Find subarray with given sum · 1 hour ago

```
    printf ("Total set bit count is %d", countSetBits(n));
    return 0;
}
```

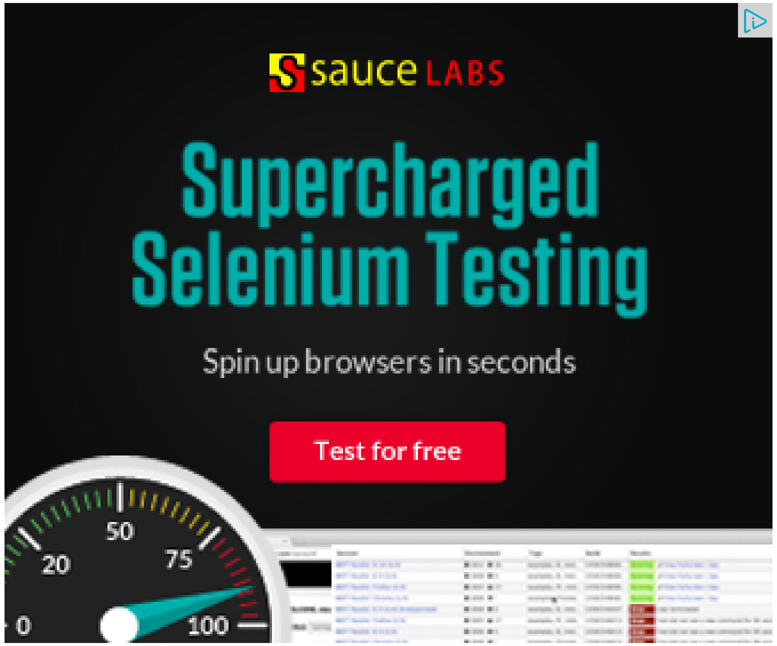
```
Total set bit count is 35
```

Time Complexity: O(Logn). From the first look at the implementation, time complexity looks more. But if we take a closer look, statements inside while loop of getNextLeftmostBit() are executed for all 0 bits in n. And the number of times recursion is executed is less than or equal to set bits in n. In other words, if the control goes inside while loop of getNextLeftmostBit(), then it skips those many bits in recursion.

Thanks to agatsu and IC for suggesting this solution.

**See this for another solution suggested by Piyush Kapoor.**

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



## Related Tpoics:

- Check if a number is multiple of 9 using bitwise operators

- How to swap two numbers without using a temporary variable?
- Divide and Conquer | Set 4 (Karatsuba algorithm for fast multiplication)
- Find position of the only set bit
- Swap all odd and even bits
- Add two bit strings
- Write your own strcmp that ignores cases
- Binary representation of a given number

9 Tweet 0 0

**Writing code in comment?** Please use **ideone.com** and share the link here.

**51 Comments** GeeksforGeeks

Sort by Newest ▾

Join the discussion…

**Deepak Gupta** • 2 months ago

Do anyone know how to do it in O(1). This solution along with all the solutions solution. I need O(1) solution for this problem:

http://www.spoj.com/problems/B...

^ | ˅ • Reply • Share ›

**being_coder** • 3 months ago

```
#define INT_SIZE 32

int main()
{

int sum=0,x,i,j;
int n=4;
for(i=0;i<int_size;i++) {="" x="1&lt;&lt;i;" for(j="1;j&lt;=n;j++)" if(j&x)="" sum++;=
bits="" is="" %d\n",sum);="" }="">
```

⌃ | ⌄ • Reply • Share ›

**Глеб Степанов** • 3 months ago

```
public static int count(int n){
int tmp = 1;
int k = 0;
int count = 0;

do {
int h = 1 << (k + 1);

count += (n + 1) / h * (1 << k);
count += ((n + 1) % h - (1 << k)) > 0 ? (n + 1) % h - (1 << k) : 0;

k++;
tmp <<= 1;
}
while (tmp <= n);

return count;
}
```

⌃ | ⌄ • Reply • Share ›

**Kai** • 4 months ago

I dought that getNextLeftmostBit() is required here, we always get (m-1) from t
(m-1) from getLeftmostBit().

⌃ | ⌄ • Reply • Share ›

**Ajay** • 6 months ago

Here is a short recursive sulution

```
#include<stdio.h>

int getLeftMostBit(int n)

{

    int m=0;

    while(n>1)

    {

        n = n >> 1;

        m++;
```

see more

⌃ | ⌄ • Reply • Share ›

**sumit** • 7 months ago

can any one tell me when the while loop inside getNextLeftmostBit() will be exe
this function it give same value of m

⌃ | ⌄ • Reply • Share ›

**manish** ➦ sumit • 6 months ago

@admin , plz some one reply for this comment , i have same problem

⌃ | ⌄ • Reply • Share ›

**Kai** ➦ manish • 4 months ago

It returns (m-1) always .. eg: for n=16, we get m = 5, from getL
getNextLeftmostBit() , 2^5 = 32 which is 'temp' value, iterate's c

Reply • Share ›

**Guest** • 9 months ago

```
#include<iostream>

#include<cmath>

int count(int n)

{

if(n==0) return 0;

int x = log2(n);

int cnt = x*(1<<(x-1)) + n - (1<<x) +="" 1;="" return="" cnt="" +="" count(n^(1<
n="17;" std::cout<<count(n);="" return="" 0;="" }="">
```

Reply • Share ›

**Guest** • 9 months ago

```
#include<iostream>

#include<cmath>

using namespace std;

int count(int n)

{

if(n==0) return 0;

int x = log2(n);

int cnt= x*(1<<(x-1)) + n - (1<<x) +="" 1;="" return="" cnt="" +="" count(n="" -=
int="" n;="" n="17;" cout<<count(n);="" }="">
```

• Reply • Share ›

**Guest** • 9 months ago

```
#include<iostream>
#include<cmath>
using namespace std;

int count(int n)
{
if(n==0) return 0;
int x = log2(n);
int cnt= x*(1<<(x-1)) + n - (1<<x) +="" 1;="" return="" cnt="" +="" count(n="" -=
int="" n;="" n="17;" cout<<count(n);="" }="">
```

• Reply • Share ›

**Rajasuba Subramanian** • 9 months ago

number of set bits when n=4 its 5 and not 6 in the first program think there is s

1 • Reply • Share ›

**Prajit** • 9 months ago

```
//fact that n&(n-1) will unset rightmost set bit

int count_bit(int n)
{
int i,tmp,sum=0;
if(n<0) { return -1; }
for(i=1;i0)
{
tmp=tmp&(tmp-1);
sum++;
}
}
return sum;
}
```

^ | ⌄ • Reply • Share ›

**sumit** • 10 months ago

```
int count_set_bits(int n)
{
int count=0;
while(n!=0)
{
n=n&(n-1);
count++;
}
```

> ```
> /* Paste your code here (You may delete these lines if not writing c
> ```

1 ^ | ⌄ • Reply • Share ›

**anshul.chauhan** • 10 months ago

A simple algo would be to count it for each bit position. It will run in constant tir
2^i pairs have occurred in ith position. For eg. if on 3rd lsd 4 pairs of 4 has occ
1s and move on. In case some remainder is left add that too if no of pairs is oc

```
[sourcecode language="JAVA"]
public static int countSetBits(int n) {
int totalBits = 0;
int i=0;
while(i<32) {
int d=(int)Math.pow(2, i);
if(d>n)break;
int q=(n+1)/d;
```

```
if(q%2==0)
totalBits+=q*d/2;
else
totalBits+=(q-1)*d/2+r;
i++;
}
return totalBits;
}
```

⌃ | ⌄ • Reply • Share ›

**Priyank Doshi** • 10 months ago

I think following approach would also work.

Idea is :

We can represent number as :

```
0000 0000
0000 0001
0000 0010
0000 0011
0000 0100
0000 0101
0000 0110
0000 0111
0000 1000
0000 1001
0000 1010
0000 1011...
```

see more

| • Reply • Share ›

pribic • 10 months ago

```
public class SetBitsUptoN {
    public static void main(String[] args) {
        int n = 20;
        for (int i = 0; i <= n; i++)
            System.out.println(i + " :" + new SetBitsUptoN().countSetI
    }

    int countSetBits(int n) {
        if (n == 0) {
            return 0;
        }
        int c = 0;
        n++;
        int bits = (int) Math.ceil(Math.log10(n) / Math.log10(2));
        for (int i = 1; i <= bits; i++) {
            double interval = Math.pow(2, i);
            double multi = interval / 2;
            double integral = Math.floor(n / interval) * multi;
            double reminderPart = (n % interval - multi <= 0) ? 0 : (
            double noOfBitinCurrentPosition = integral + reminderPart
            c += noOfBitinCurrentPosition;
        }
        return c;
    }
}
```

^ | ˅ • Reply • Share ›

Chaude ho rahe ho • 11 months ago

In method 1 for given example output should be 5...its given as 6.

Reply • Share ›

**Nishant Kumar** • a year ago

Algo of method 2 is good but in the code of method 2 getNextLeftmostBit() is u
_countSetBits() are used if we can achieve same only by _countSetBits().

```
 int myTotalSetBitCount(int n){
        if(n==0)
                return 0;
        int m = getLeftmostBit(n);

        if(n==((1<<(m+1))-1))
                return (m+1)*(1<<m);

        n = n - (1<<m);

        return (n+1) + myTotalSetBitCount(n) + m*(1<<(m-1));
}
```

Reply • Share ›

**Hanish Bansal** Nishant Kumar • 11 months ago

Exactly !!

Reply • Share ›

**Ashutosh** Nishant Kumar • a year ago

Absolutely.

Reply • Share ›

**rahul23** • a year ago

@geeksforgeeks...Why are you using getNExtleft fxn...whose while loop will n

```
/* Paste your code here (You may delete these lines if not writing c
```

^ | ⌄ • Reply • Share ›

**cfchou** • a year ago

Try this in haskell with acceleration through meomoization. It might be in the s
haven't checked.

```
[sourcecode language="Haskell"]
import Data.Array

allOnes :: Int -> Int
allOnes n
| n <= 0 = 0
| otherwise = let k = log2Floor n
arr = mkArray k
in allOnes' n arr

allOnes' :: Int -> Array Int Int -> Int
allOnes' 0 _ = 0
allOnes' n arr = let k = log2Floor n
d = n - (2^k - 1)
in if d == 0 then arr ! k
else (arr ! k) + d + (allOnes' (d - 1) arr)

log2Floor :: Int -> Int
log2Floor n = floor $ logBase 2 (fromIntegral n)

mkArray :: Int -> Array Int Int
mkArray k = array (0, k) [ (i, f i) | i <- [0..k] ]
where f 0 = 0
f 1 = 1
```

• Reply • Share ›

**cfchou** cfchou • a year ago

better formatted

https://gist.github.com/cfchou...

• Reply • Share ›

**san** • a year ago

If the input number is of the form 2^b -1 e.g., 1,3,7,15.. etc, the number of set

this is incorrect

if number is of the form 2^b -1 , the number of set bits is b.

eg

1 0001 (2^1 - 1) 1

3 0011 (2^2 - 1) 2

7 0111 (3^2 - 1) 3

15 1111 (4^2 - 1) 4

• Reply • Share ›

**hero** • 2 years ago

```
count=0;
for (x=1;x<=n;x++)
{
while(x)
{
x&(x-1);
count++;
}
}
return count;
```

• Reply • Share ›

**White Tiger** • 2 years ago

```
 int findSetBits(int n)
{
        int i=0,k=n,count=0,total=0;
        char arr[32];
        while(k)
        {
                arr[i++]=(k&1)+48;
                k=k>>1;
        }
        arr[i--]='&#092&#048';
        while(i>=0)
        {
                if(arr[i]==49)
                {
                        total+=(i+2*count)*pow(2,i-1)+1;
                        count++;
                }
                i--;
        }
        return total;
}
```

Time Complexity: O(log n)

^ | ⌄ • Reply • Share ›

**Sourabh mehrotra** • 2 years ago

```
 #include <stdio.h>
#include <math.h>
```

```
{
    int num=17,i,sum=0,aux,b;
    b=countbits(num);
    b--;
    sum=b*pow(2,b-1);
    aux=1<<b;
    printf("%d\n",aux);
    while(aux<=num)
    {
        sum+=count1(aux);
        aux++;
    }
    printf("%d",sum);
    getchar();
```

see more

^ | ˅ • Reply • Share ›

**Anil** • 2 years ago

Please sent me steps how to count bits ?

^ | ˅ • Reply • Share ›

**lomash** • 2 years ago

for this problem-we can do the following approach
1)count the no of bits in the number-
while(n)
{
n=n/2;
count++;
}
2) return count+pow(2,count-1);

```
/* Paste your code here (You may delete these lines if not writing c
```

• Reply • Share ›

**LOMASH** lomash • 2 years ago

int countsetbitsupton(int n)
{
y=countsetbitsuptonn(n);
int x=n-2^y-1;
return(countsetbitsupton(2^n-1)+x+countsetbitsupton(x));
}

PLEASE TELL ME THE COMPLEXITY OF THIS

```
/* Paste your code here (You may delete these lines if not wr
```

• Reply • Share ›

**lomash** lomash • 2 years ago

sorry i have written the wrong code..

• Reply • Share ›

**Ravi aka smash** • 2 years ago

```
 #include<stdio.h>
#include<stdlib.h>
#include<math.h>
int sum=0;
unsigned int getleftmostbit(unsigned int n)
{
    int k=0;
    while(n>1)
    {
        k++;
        n=n>>1;
    }
    return k;
}
unsigned int get(unsigned int n)
{
    int m=0;
    m=getleftmostbit(n);
```

see more

^ | ˅ • Reply • Share ›

**Oar** ➦ Ravi aka smash • 2 years ago

This one is better than the code posted by the author. 'getNextLeftmos
'getLeftmostBit'

^ | ˅ • Reply • Share ›

**krishna** • 2 years ago

Can any one please put the above mathematical logic in words, so that it woul

^ | ˅ • Reply • Share ›

**krishna** • 2 years ago

For the method 1 : the running time must be O(n), since finding numbers of or
given size integer

^ | ˅ • Reply • Share ›

**Narendra** • 2 years ago

#include

unsigned count_set_bits(unsigned num);

```
{
unsigned i, n, cnt;

cnt = 0;

printf("Enter the range: ");
scanf("%u", &n);

for (i = 1; i > 1) & 0x55555555);
num = (num & 0x33333333) + ((num >> 2) & 0x33333333);
num = (num & 0x0f0f0f0f) + ((num >> 4) & 0x0f0f0f0f);
num = (num & 0x00ff00ff) + ((num >> 8) & 0x00ff00ff);
num = (num & 0x0000ffff) + ((num >> 16) & 0x0000ffff);

return num;
}
```

^ | ⌄ • Reply • Share ›

**Narendra** ➜ Narendra • 2 years ago

Sorry i paste here wrong code.

^ | ⌄ • Reply • Share ›

**Piyush Kapoor** • 2 years ago

A simple solution , using the fact that for the ith least significant bit, answer wil where X = N%(2^i) - (2^(i-1)-1) iff N%(2^i)>=(2^(i-1)-1)

```
int getSetBitsFromOneToN(int N){
   int two = 2,ans = 0;
   int n = N;
   while(n){
       ans += (N/two)*(two>>1);
       if((N&(two-1)) > (two>>1)-1) ans += (N&(two-1)) - (two>>1)+1;
       two <<= 1;
```

```
        n >>= 1;
    }
    return ans;
}
```

1 ⌃ | ⌄ • Reply • Share ›

**kartik** ➦ Piyush Kapoor • 2 years ago

@Piyush Kapoor: Thanks for sharing your code. Your method looks be
It will helpful for us if you provide more explanation so that we can add

⌃ | ⌄ • Reply • Share ›

**Piyush Kapoor** ➦ kartik • 2 years ago

Consider the ith least significant bit(1 based indexing) for numb
a period equal to 2^i.And in the period , first half of the values ar
example :-
For numbers from 0 to 7,(0 will contribute nothing so no effect)
000
001
010
011
100
101
110
111
1st least significant bit = 01010101 Period=2
2nd least significant bit = 00110011 Period=4
3rd least significant bit = 00001111 Period=8
and so on.
So for the ith least significant bit ,answer will be (N/Period)*(Ha
Half of Period Size)

The second term will only be taken when N%Remainder is grea
Size.
Also , N%(2^i) can be written as N&(2^i - 1)

```
/* Paste your code here (You may delete these lines if
```

1 ^ | ⌄ • Reply • Share ›

**Hanish Bansal** ➦ Piyush Kapoor • 11 months ago

Really nice !!

1 ^ | ⌄ • Reply • Share ›

**Piyush Kapoor** ➦ Piyush Kapoor • 2 years ago

A correction :
Second Term in the answer is
N%(2^i - 1) - Half of Period Size + 1

which is only taken when N%(2^i - 1) is greater than or e

1 ^ | ⌄ • Reply • Share ›

**User** ➦ Piyush Kapoor • 2 years ago

@Piyush Kapoor: I couldnt understand this part.
if((N&(two-1)) > (two>>1)-1) ans += (N&(two-1)) - (two>

Please explain on this more.

1 ^ | ⌄ • Reply • Share ›

**Abhijeet Sinha** ➦ Piyush Kapoor • 2 years ago

awesome :-)

^ | ⌄ • Reply • Share ›

**kartik** ➦ Piyush Kapoor • 2 years ago

Thanks Piyush. We will soon add it to the original post.

^ | ˅ • Reply • Share ›

**Piyush** • 2 years ago

A simple solution , using the fact that for the ith least significant bit, answer wil where X = N%(2^i) - (2^(i-1)-1) if N%(2^i)>=(2^(i-1)-1)

```
 int getSetBitsFromOneToN(int N){
    int two = 2,ans = 0;
    int n = N;
    while(n){
        ans += (N/two)*(two>>1);
        if((N&(two-1)) > (two>>1)-1) ans += (N&(two-1)) - (two>>1)+1;
        two <<= 1;
        n >>= 1;
    }
    return ans;
}
```

^ | ˅ • Reply • Share ›

**indra kumar** ➦ Piyush • 10 months ago

very good method

^ | ˅ • Reply • Share ›

**dipendra** ➦ Piyush • a year ago

could not get the second part

if((N&(two-1)) > (two>>1)-1) ans += (N&(two-1)) - (two>>1)+1;

can you explain it....

^ | ˅ • Reply • Share ›

**Chiranjeev Kumar** ➦ Piyush • 2 years ago

wowwwwwwww :)
awesome :-)

1 ⌃ | ⌄ • Reply • Share ›

Load more comments

✉ Subscribe   Add Disqus to your site