Write an Efficient C Program to Reverse Bits of a Number

Method1 – Simple

Loop through all the bits of an integer. If a bit at ith position is set in the i/p no. then set the bit at (NO_OF_BITS – 1) – i in o/p. Where NO_OF_BITS is number of bits present in the given number.

```
/* Function to reverse bits of num */
unsigned int reverseBits(unsigned int num)
{
    unsigned int  NO_OF_BITS = sizeof(num) * 8;
    unsigned int reverse_num = 0, i, temp;

    for (i = 0; i < NO_OF_BITS; i++)
    {
        temp = (num & (1 << i));
        if(temp)
            reverse_num |= (1 << ((NO_OF_BITS - 1) - i));
    }

    return reverse_num;
}

/* Driver function to test above function */
int main()
{
    unsigned int x = 2;
    printf("%u", reverseBits(x));
    getchar();
}
```

Above program can be optimized by removing the use of variable temp. See below the modified code.

```
unsigned int reverseBits(unsigned int num)
{
    unsigned int  NO_OF_BITS = sizeof(num) * 8;
    unsigned int reverse_num = 0;
    int i;
    for (i = 0; i < NO_OF_BITS; i++)
    {
        if((num & (1 << i)))
           reverse_num |= 1 << ((NO_OF_BITS - 1) - i);
   }
    return reverse_num;
}
```

Time Complexity: O(log n)
Space Complexity: O(1)

Method 2 – Standard
The idea is to keep putting set bits of the num in reverse_num until num becomes zero. After num becomes zero, shift the remaining bits of reverse_num.

Let num is stored using 8 bits and num be 00000110. After the loop you will get reverse_num as 00000011. Now you need to left shift reverse_num 5 more times and you get the exact reverse 01100000.

```
unsigned int reverseBits(unsigned int num)
{
    unsigned int count = sizeof(num) * 8 - 1;
    unsigned int reverse_num = num;

    num >>= 1;
    while(num)
    {
       reverse_num <<= 1;
       reverse_num |= num & 1;
       num >>= 1;
       count--;
    }
    reverse_num <<= count;
    return reverse_num;
}

int main()
{
    unsigned int x = 1;
    printf("%u", reverseBits(x));
```


Popular Posts

```
    getchar();
}
```

Time Complexity: O(log n)
Space Complexity: O(1)

Method 3 – Lookup Table:
We can reverse the bits of a number in O(1) if we know the size of the number. We can implement it using look up table. Go through the below link for details. You will find some more interesting bit related stuff there.

http://www-graphics.stanford.edu/~seander/bithacks.html#BitReverseTable

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

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

26 Comments GeeksforGeeks

Sort by Newest

rcrox • 2 months ago

There is a major flaw in first solution.
ith bit should be NO_OF_BITS +1 -i from reverse
rather than NO_OF_BITS -1-i
Ur solution will give a garbage value

Atul • 5 months ago

```
#include <stdio.h>
int main()
{
unsigned int num=67;
int temp1,temp2;
int temp;
int j,i=31;
```


Recent Comments

Abhi You live US or India?
Google (Mountain View) interview · 28 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 · 2 hours ago

printf("Number is =%d\n",num);

while(i>=0)
{
temp = num&(1<<i); sizeofnum++;="" if(temp)="" break;="" i--;="" }="" sizeofn
printf("lenght="" of="" number="%d",sizeOfNum);" i="0;" j="sizeOfNum;" while
{
temp1=num&(1<<i); temp2="num&(1<<j);" if((temp1="=0" &&="" ter
temp2="=0))" {="" num="num^(1<<i);" num="num^(1<<j);" }="" i++;=""
number="" is="%d\n",num);" return="" 0;="" }="">

• Reply • Share ›

Shradha Agrawal • 9 months ago

unsigned reverse(unsigned num).
{
unsigned reverse_num=0;.
while(num!= 0).
{.
if(num & 1).
reverse_num = 2*reverse_num + 1;.
else.
reverse_num = 2*reverse_num;.
num >>= 1;.
}.
return reverse_num;.

}

6 • Reply • Share ›

hemanthreddy • 9 months ago

```
#define INT_SIZE 32
```

```
void reverse_bits(unsigned int num)
{
        int i=INT_SIZE-1;
        unsigned int rev_num=0;
        while(i>=0 && num)
        {
                if(num & 1)
                        rev_num |= (1<<i);
                num>>=1;
                i--;
        }
        printf("\nreversed number: %u\n",rev_num);
}
```

3 ▲ | ▼ • Reply • Share ›

olive • 10 months ago

wont ~no suffice

▲ | ▼ • Reply • Share ›

Hanish • a year ago

In method 2, is it necessary to initialise reverse_num = num even though we d
taken from n. and why do we check the first bit of n outside the while loop.is da

Is the following code correct or are there some corner cases missing.Please e

unsigned int reverseBits(unsigned int num)
{
unsigned int count = sizeof(num) * 8 ;
unsigned int reverse_num = 0;

```
{
reverse_num <>= 1;
count--;
}
reverse_num <<= count;
return reverse_num;
}
```

Reply • Share ›

Hanish ➔ Hanish • a year ago

Sorry,there was a typing error . It should be :

```
while(num)
{
reverse_num <>= 1;
count--;
}
```

Reply • Share ›

Ashish • a year ago

Here is one more way, although standard only with n/2 complexity.

```
int bitreverse(int x)
{
        unsigned int lmask = 0x80000000;
        int rmask = 0x01;

        while(lmask>rmask)
        {
                if(!(((mask1&x) != 0 && (mask2&x) != 0) || ((mask1&x)=
                {
                        x=x^mask1;
```

```
                x=x^mask2;
            }
            mask1>>=1;
            mask2<<=1;
        }

        return x;
}
```

• Reply • Share ›

Ashish Ashish • a year ago

Corrected with variable names:

```
int bitreverse(int x)
{
        unsigned int lmask = 0x80000000;
        int rmask = 0x01;

        while(mask1>mask2)
        {
                if(!(((lmask &x) != 0 && (rmask&x) != 0) || ((
                {
                        x=x^lmask;
                        x=x^rmask;
                }
                lmask >>=1;
                rmask <<=1;
        }

        return x;
}
```

ikram • 2 years ago

```
#include
#include
#define BIT 5
void swap(char *, char *);
int main()
{
char reverse[33];
int num=18,i,a=1;

for(i=BIT-1;i>=0;i--)
{
if(num & a)
reverse[i]='1';
else
reverse[i]='0';
a=a<<1;
}
reverse[BIT]=&#039&#039;
```

see more

Optimus • 2 years ago

Method 3 won't work for negative integers.

Tp • 2 years ago

```
#include<stdio.h>
unsigned int swapBits(unsigned int x, unsigned int i, unsigned int j)
{
        unsigned int left=((x>>i)&1);
        unsigned int right=((x>>j)&1);
        if(left^right)
                x^=((1U<<i)|(1U<<j));
        return x;
}
main()
{
        unsigned int x=1;
        int n = sizeof(x)*8;
        int i=0;
        for(i=0;i<n/2;i++)
                x=swapBits(x,i,n-i-1);
        printf("after reversing:%u\n",x);
}
```

Reply • Share ›

Bharath • 3 years ago

How is the time complexity of method (2) log(n)?

Reply • Share ›

nicks • 3 years ago

why the complexity in the first one is **O(logn)** it should be **O(n)** ??

1 Reply • Share ›

vendetta • 3 years ago

```
unsigned int reverseBits(unsigned int num)
{
    unsigned int i=0;
    i--;
    return (num^i);
}
```

its O(1)!!! we can also use ~num

Reply • Share ›

Venki ➚ vendetta • 3 years ago

@vendetta, it is bit reversal not complimenting every bit. Please check

Reply • Share ›

seeker7 • 4 years ago

hi can anyone pls elaborate on the complexity,why is it o(log n) for method 1?? thanks

Reply • Share ›

Shekhu ➚ seeker7 • 4 years ago

I think the correct time complexity should be O(log(MAX_UNSIGNED_I
MAX_UNSIGNED_INT is maximum possible value of unsigned integer.

Comments are welcome.

Reply • Share ›

Venki • 4 years ago

Another solution here, in the order of log(N)

http://math-puzzles-computing....

Reply • Share ›

Venki Venki • 4 years ago

I am providing the working code

```
unsigned __int32 reversing_bits(unsigned __int32 input)
{
    // complixity O(log [no.of.bits]) = O(1)
    // On 32 bit machines it takes 5 steps (logical)

    // Step 1
    // Mask bit positions 0, 2, 4... shift LEFT this masked num
    // Mask bit positions 1, 3, 5... shift RIGHT this masked nu
    input = (input & 0x55555555) <<  1 | (input & 0xAAAAAAAA) >

    // Step 2
    // Mask bit positions 01, 45, 89... shift LEFT this masked
    // Mask bit positions 23, 67... shift RIGHT this masked num
    input = (input & 0x33333333) <<  2 | (input & 0xCCCCCCCC) >
```

see more

Reply • Share ›

sarath Venki • 3 years ago

really smart...!!

Reply • Share ›

Dheeraj Venki • 4 years ago

please provide pseudo code / C code for this solution?

Reply • Share ›

geeksforgeeks • 4 years ago

@game & hunny: Thanks very much for pointing this out. We have corrected t

Reply • Share ›

hunny • 4 years ago

In the last few codes, the problem is the order of

reverse_num |= num & 1;
reverse_num <<= 1;

you do an extra shift left, which should not be done.

Kindly look into it.

Reply • Share ›

hunny • 4 years ago

The reversal of bits of 2 should give 1073741824, but your method gives -2147

Reply • Share ›

game • 4 years ago

I feel that the order the following lines should be reversed in all the codes.
"reverse_num |= num & 1;
reverse_num <<= 1;"

Actually the last line right shifts the reverse_num, at the last iteration, so the re
shift which it should not have.

1 Reply • Share ›